VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Generování uživatelského rozhraní z entitního modelu pro prostředí Eclipse

User Interface Generation from Entity Model in Eclipse Enviromental

2014 Roman Ollé

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Roman Ollé**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Generování uživatelského rozhraní z entitního modelu pro prostředí Eclipse
User Interface Generation from Entity Model in Eclipse Environment

Zásady pro vypracování:

Cílem práce je vytvoření pluginu pro Eclipse, jehož hlavní funkcionalitou je vygenerování kompletního uživatelského rozhraní (JSF stránky) z vybraných tříd entit. Vytvořené webové stránky každé entity budou obsahovat všechny základní typy operací (select, insert, update, delete). Architektura pluginu umožní jeho rozšíření o generování uživatelského rozhraní pro jiné zvolené rámce.

Práce bude obsahovat:
1. Rozbor problematiky Eclipse pluginu a JSF stránek.
2. Návrh a implementaci pluginu (jeho funkcionalitu, GUI wizard a nabídky navigace v Eclipse).
3. Dokumentaci k vyvíjenému systému.
4. Výsledky testu funkčnosti na různých typech modelů entit.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/
[3] William Crawford, Jonathan Kaplan: J2EE Design Patterns. O'Reilly Media. 2003. ISBN:978-0-596-00427-9.

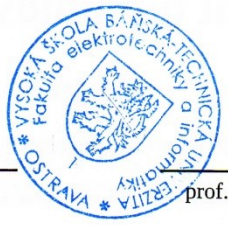
Dále podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: 6.května 2014 .....

Roman Ollé

Poděkování

Rád bych poděkoval svému vedoucímu mojí diplomové práce panu Ing. Davidu Ježkovi za čas pro moje dotazy, odbornou pomoc a konzultace při vytváření této práce.

Abstrakt

V této diplomové práci se řeší vytváření pluginu pro vývojové prostředí Eclipse s platformou EE. Tento plugin generuje kompletní webové stránky z vybraných JPA entit. Vygenerované stránky můžou být buď typu JSP nebo JSF. V práci jsou popsány všechny nejdůležitější technologie, které byly při vývoji použité. Důležitou částí je podrobná analýza a návrh celého pluginu, kde se řeší různé typy entit a jejich atributy, a jaký mají vliv na výsledné webové stránky. Práce obsahuje popis Eclipse wizardu a jeho vytvoření, dále jsou rozebrány jednotlivé typy webových stránek, jejich výsledný kód a rozdíly mezi nimi. Na konci je umístěno shrnutí výsledné práce s popisem nedostatků a návrhem na další rozšíření. K práci je přiložen manuál pro uživatele s popisem obsluhy a možnostech vygenerovaných webových stránek.

Klíčová slova

webové stránky, plugin, wizard, Eclipse, Java EE, JSF, JSP

Abstract

In this diploma thesis is solved to creation plug-in for Eclipse EE environmental. This plug-in generates complete web pages from selected JPA entities. Generated pages might be either JSP type or JSF. This thesis describes all of the most important technologies used in development. The important part is detail analysis and design of plug-in which solve various types of entities, their attributes, and their effect on the result web pages. Thesis contains a description of the Eclipse wizard and its creation. Next is analysis of all web pages types which are used in wizard, and their final code and differences between them. There is summary of the final work on the end which describes lacks and proposals for further extension. The user guide for description of the controlling and all possibilities of the generated web pages are attached to thesis.

Key words

web pages, plug-in, wizard, Eclipse, Java EE, JSF, JSP

Seznam použitých symbolů a zkratek

API	Application Programming Interface
AST	Abstract Syntax Tree
ASP	Active Server Pages
CSS	Cascading Style Sheets
DAO	Data Access Object
DOM	Document Object Model
EJB	Enterprise Java Bean
EL	Expression Language
EPL	Eclipse Public Licence
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
ISBN	International Standard Book Number
IT	Information Technology
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
JAR	Java Archive
Java EE	Java Enterprise Edition
Java SE	Java Standard Edition
JCP	Java Community Process
JDT	Java Development Tools
JPA	Java Persistence API
JPE	Java Profesional Edition

JSF	Java Server Faces
JSP	Java Server Pages
JSTL	JSP Standard Tag Library
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
POJO	Plain Old Java Object
PHP	Hypertext Preprocessor, dříve Personal Home Page
RCP	Rich Client Platform
SQL	Structured Query Language
SWT	The Standard Widget Toolkit
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	eXtensible Markup Language

Seznam obrázků

Seznam příkladů

Seznam tabulek

Obsah

1 Úvod

Pojem webové stránky zná v dnešní době téměř každý člověk této planety, a to díky rozsáhlé multimediální technologii zvané "internet", který obsahuje miliardy existujících webových stránek, jejichž cílem je uchovávat, sdělovat, poskytovat informace.

Ale jak se tyto webové stránky vlastně rodí? Odpověď je jednoduchá, weby jsou převážně tvořeny skupinou IT odborníků, kteří přesně vědí, jak je vytvářet, patří tam například návrhář, programátor, apod.

Samotné webové stránky můžou být od jednoduchých až po složité, můžou být statické či dynamické, které využívají různé datové zdroje, jako je například relační databáze, textové soubory, XML soubory, atp. To znamená, že u dynamických stránek může uživatel měnit v čase obsah webových stránek, kde vložená data se ukládají do výše uvedeného datových zdrojů. Čím bude větší množství údajů na uložení a čím více vazeb bude mezi jednotlivými údaji, tím vzniká velká složitá struktura těchto webových stránek. Z této složitosti webu, pak vychází i náročnost na jeho vytvoření, takže čím bude web složitější, tím bude obtížnější ho vytvořit. Tím ale nezůstaneme pouze na obtížnosti vývoje, tento aspekt se odráží na další vlastnost, kterou je čas potřebný pro vývoj webu, to znamená, že vytvoření větších projektů má tendenci trvat delší dobu, a jak je nám všem dobře známo, čas jsou peníze.

Vyšlo nám tedy, že vytvoření webových stránek nebude moc levná záležitost, při nějakém větším projektu se může cena vyšplhat hodně vysoko a to nikdo nechce, hlavně ne obyčejný uživatel. A proto se hledá způsob, jak celý tento vývoj zjednodušit a urychlit, a přesně o tohle půjde v této diplomové práci.

Cílem je tedy zjednodušení a urychlení samotného vývoje webových stránek, z toho plyne, podle toho, co bylo výše uvedené, že by měl být snížený čas a finance potřebné na vývoj. Hlavní myšlenkou je automatizované vytváření kompletních webových stránek, podle schématu návrhu celého datového modelu (relační databáze, XML, atd.), což znamená, že se bude převádět datový model ihned na webové stránky, bez nutnosti velkého programování.

Výsledkem celé této práce pak bude plugin pro Eclipse J2EE, který díky svému jednoznačnému uživatelskému rozhraní bude jednoduše použitelný i pro méně zkušené osoby. Pro úspěšné vygenerování stránek bude zapotřebí pouze entitní model, který si uživatel s přispěním základních znalostí programovacího jazyku Java dokáže vytvořit sám. Popřípadě existuje ještě druhá možnost, a tou je znalost relačních databází nebo alespoň tvorbu datového modelu, protože lze pomocí jiného existujícího pluginu tento model převést do javovského entitního modelu.

Výstupem z pluginu budou funkční webové stránky zobrazeny přes jednoduché uživatelské rozhraní. Samotný vygenerovaný kód bude napsaný v Javě, to znamená JSF/JSP stránky se sadou

potřebných zdrojových java kódů. Uživatel pak bude mít na stránkách možnost vkládání nových dat do databáze, následně je může měnit, popřípadě smazat. Vzhled stránek bude defaultně příliš jednoduchý, žádný složitý design s obrázky, menu atp., ale pouze prostý text zobrazující data. Takže s troškou znalostí HTML a JSF/JSP si pak lze stránky přizpůsobit podle své fantazie.

Výsledné webové stránky můžou například pomoct vývojářům při tvorbě stránek, a to že jim zkrátí čas vývoje, stačí jim dodělat vzhled stránek nebo udělat ve vygenerovaných stránkách nějaké úpravy ve zdrojovém kódu jako je nezobrazení identifikátoru, popřípadě sloučit několik těchto stránek, neboli informací, do jedné stránky, třeba jako detail zboží s nějakými novinkami nebo nějaké akce apod. Zkušenější uživatele si můžou pro jednotlivé stránky nebo celý webu vytvořit práva pro různé administrátory a typy uživatelů. Výsledné stránky můžou taky sloužit jen pro různé testování.

2 Technologie

2.1 Eclipse

Eclipse je open source platforma určena především pro vývoj aplikací, tento typ aplikací se obecně nazývá vývojové prostředí. Prvotní verze 1.0 projektu Eclipse vznikl v roce 2001 uvolněním kódu firmou IBM pod licencí EPL. Odhad příspěvku firmy IBM pro vývoj tohoto open source nástroje se odhaduje kolem 40 miliónů dolarů. Eclipse byl primárně určený pro programování v jazyce Java. Pro vzhled tohoto projektu byl vyvinutý Framework SWT, který je nativní vzhled aplikací pro všechny platformy, kde je SWT podporován, to znamená, že využívá nativního kódu operačního systému. Později vznikl nový Framework nazývaný Swing, který využívá pouze služeb JVM, čímž je zaručena lepší přenositelnost.

Pozdější verze pak přijaly nový standard OSGi R4, čímž získaly nové vlastnosti pro vývojáře, jelikož balíčky OSGi přinášejí možnost snadné rozšiřitelnosti aplikace o novou funkcionalitu. Tím lze dynamicky rozšířit Eclipse o nové funkce podle potřeby uživatele, díky tomu se Eclipse stává flexibilní. Tato vlastnost přináší možnosti rozšířit funkcionalitu softwaru, což může vést k podpoře i jiných programovacích jazyků jako je třeba C++, Python, Ruby nebo PHP. Ale taky pro práci s jinými než Java soubory, jako můžou být například HTML či XML soubory. Existují i pluginy, pro grafické návrhy aplikací, třeba pro návrh desktopové aplikace, které v základní verzi Eclipse nejsou obsaženy. Existují celá řada subprojektů Eclipse, které jsou zaměřeny na různé odvětví programování, jako je třeba J2EE, J2ME, ale třeba i pro výše zmíněný vývoj v C++, PHP apod. [1]

2.2 Plugin

Pluginy jsou programy, které jsou instalovány do jiných aplikací. Pluginy těmto aplikací přinášejí nějakou novou funkcionalitu. Samostatně pluginy pracovat nemohou, musí být obsaženy v nějaké přípustné aplikaci, který tento plugin dokáže přijmout. Slovo plugin je pojem přejatý z angličtiny, v češtině existuje pro něho přeložené pojmenování "zásuvný modul". V angličtině dále pro něho existuje více názvů jako je plug-in, extension, add-on.

Pluginy se v dnešní době nejčastěji používají ve spojení s internetovými prohlížeči, které mají většinou vytvořenou své vlastní webové stránky, jinak řečeno skladiště, kde mají k dispozici tisíce různých pluginů pro rozšíření jejich prohlížeče. Pluginy nejsou vytvářeny jen pro prohlížeče, ale taky pro plno hudebních, grafických, textových, vývojových a plno dalších aplikací. Pro tyto aplikace jsou pluginy dostupné na jejich oficiálních webových stránkách, nebo je lze stáhnout z různých webových stránek, ale ne všechny tyto pluginy jsou dostatečně ohodnocené ostatními uživateli, čímž roste riziko nebezpečí pro uživatelův počítač jako třeba krádeží soukromých dat, a podobně. Ale taky se nabízí možnost, že si uživatel může vytvořit

vlastní plugin, který potřebuje nebo aby mu usnadnil práci s aplikací. Pro vlastní plugin musí mít uživatel dostatečné znalosti v tomto oboru. [2]

2.3 Eclipse plugin

Samotná platforma Eclipse slouží vývojářům především pro vývoj Java aplikací. Tento software je zaměřený na budování celého IDE, to znamená, že na jádro Eclipse se nabaluje plno pluginů, čímž je ve výsledku vytvořené kompletní vývojové prostředí vhodné pro vývoj nových aplikací. Základní verze Eclipse Standard obsahuje na začátku pouze základní vlastnosti Javy SE, jako jsou kompilátor, debugger, atp. Pro další funkce jako jsou grafické nástroje pro správu různých souborů nebo doplněk pro jednodušší vývoj desktopových aplikací je zapotřebí nainstalovat potřebný plugin, který plní požadovanou funkcionalitu. Eclipse lze takto rozšířit o funkcionalitu, které umožní uživateli třeba návrh UML, psaní HTML kódů, XML dokumentů, ale i například různé generování a vytváření specifických souborů. Existují i různé subprojekty, které se liší od základního Eclipse Standard tím, že v počáteční verzi obsahují oproti Eclipse Standard různé sady pluginy. Na oficiálních stránkách jsou k dispozici například Eclipse pro vývoj J2EE, J2ME, RCP, Eclipse se sadou pro testování, ale taky Eclipse pro vývoj aplikací v jazyce C++, Python, Ruby nebo třeba PHP, a mnoho dalších [1].

2.3.1 Cesty připojení pluginu

Plugin se dá k wokbench připojit dvěma způsoby, buď přes action a nebo pomocí command. Oba tyto typy se nastavují v plugin.xml pomocí přesně definovaných tagů. U typu action, je uživatelské rozhraní spolu se zpracováním pevně svázané a nemůžou být nijak od sebe rozdělené. Takže pokud chceme mít stejnou akci na různých místech (jako je menu, reakce na pravé tlačítko, událost v nějakém jiném dialogu, apod.), tak musíme mít stejnou, v některých případech může mít podobnou, konfiguraci rozšiřujících bodů (přejato z angličtiny z extension point) uložených pro každou akci na jiných místech. Některé konfigurace rozšiřujících bodů pro stejnou akci v různých místech, nemusí být shodné, protože některé místa očekávají v konfiguraci jiné vstupní parametry. Takže pokud máme stejnou akci na několika místech, tak musí mít každý tento bod svojí vlastní konfiguraci, která není na ničem jinem závisle kromě sebe, a pokud budeme chtít v pluginu něco změnit, například text, ikonku, apod., tak nastává problém, jelikož změnu musíme provést ve všech rozšiřujících bodech, což může být někdy obtížné [3].

Kvůli těmto nedostatkům připojení pluginu do aplikace pomocí action byl vyvinutý framework nazvaný command, který eliminuje výše zmíněné nevýhody u action. Na začátku se vytvoří samotný odkaz na plugin pomocí tagu command, který si pak můžou rozšiřující bod libovolně načítat, takže tyto body můžou odkazovat na stejný plugin, ale nastavení pluginu je umístěno pouze na jednom místě. Na definovaný command pak můžeme připojit jeho obrázek nebo handler pro zpracování, to celé znamená, že tohle rozšiřující nastavení je vloženo jen jednou, čímž se zjednoduší následná manipulace s pluginem a jeho všemi rozšiřujícími body [4].

2.4 Wizard

Wizard je nástroj, který řídí samotné chování vyvolané nějakou událostí, jenž ho volá. Wizard obsahuje jednoduchou sekvence kroků, aby rozložil složitější úkoly na části, tudíž zjednodušil méně zkušeným uživatelům práci v aplikacích. Principem je, aby uživatel viděl postupně za sebou kroky, které musí splnit, aby dosáhl požadovaného výsledku, dá se říct, že je to takové vodítko neboli šablona vedoucí uživatele k nějakému rozumnému cíli. Pro zjednodušení práce uživateli se využívá grafického uživatelského rozhraní, které obsahuje formulářové potřebná vstupní políčka, jejímž vyplněním získá wizard potřebná uživatelská data, aby mohl vykonat žádaný úkol.

Někteří zkušenější uživatele úmyslně nepoužívají při svém návrhu wizarda, protože wizard ne vždycky nabízí nejlepší řešení daného problému a nabídku pro své přizpůsobení, proto raději místo wizarda zvolí prázdný dokument, který si vytvoří dle vlastní potřeby [5].

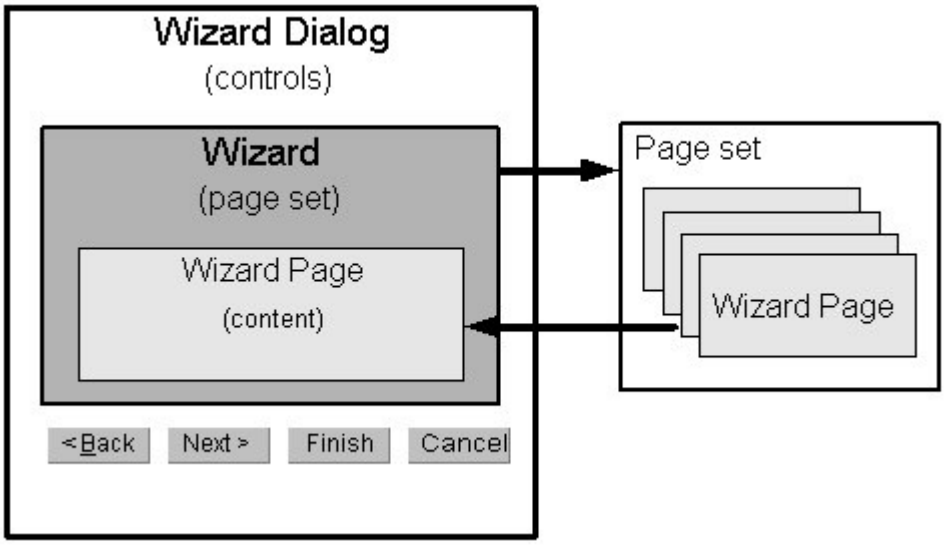
2.4.1 Vlastní pluginu v Eclipse

Zpracování události pluginu je možné provést několika způsoby, ten nejjednodušší je plugin připojit na existující handler, který tuto událost zpracuje a udělá svojí práci. Kromě zpracování handlerem existuje i možnost pracovat s wizardem [6].

Vývoj vlastní wizard v Eclipse se skládá z několika základních částí. Prvním bodem je, jak vlastní wizard spustit. K tomu budeme potřebovat výše zmíněné cesty připojení pluginu, takto připojíme plugin na wizarda, který bude řídit jeho chování.

Samotný wizard představuje grafické uživatelské rozhraní, kde na nejvyšší úrovni je třída wizard dialog. Tato třída se stará o správu stránek wizardu a definuje potřebné tlačítka, tudíž poskytuje a spravuje 3 hlavní tlačítka pro všechny stránky, kterými jsou Další, Předchozí a Dokončit. Tyto tlačítka nastavuje na stav povoleno nebo zakázáno a to podle získaných informací z aktuální stránky wizardu. O úroveň níže je třída Wizard, jež řídí celkové chování wizardu, implementuje mnoho detailu samotného chování wizardu. Například co se má stát, když se wizard dokončí nebo jaké všechny stránky wizard obsahuje, a podobně.

Posledním důležitou částí wizardu jsou samostatné stránky, které zobrazují obsah wizardu, tyto stránky by měli být rozděleny podle předem definované sekvence potřebných kroků, určitě by neměla jedna stránka obsahovat nadměrné množství komponent a naopak nemělo by existovat hodně komponent s miniaturním obsahem. Každá stránka reaguje na událost podle jeho obsahu, kdy je stránka připravena pro dokončení. To znamená, přes událost povolí tlačítko Další, pokud existuje další stránka za aktuální stránkou, popřípadě tlačítko Dokončit, když je poslední stránkou wizardu [7]. Celá tato struktura wizardu je zobrazena na Obrázek č.1.

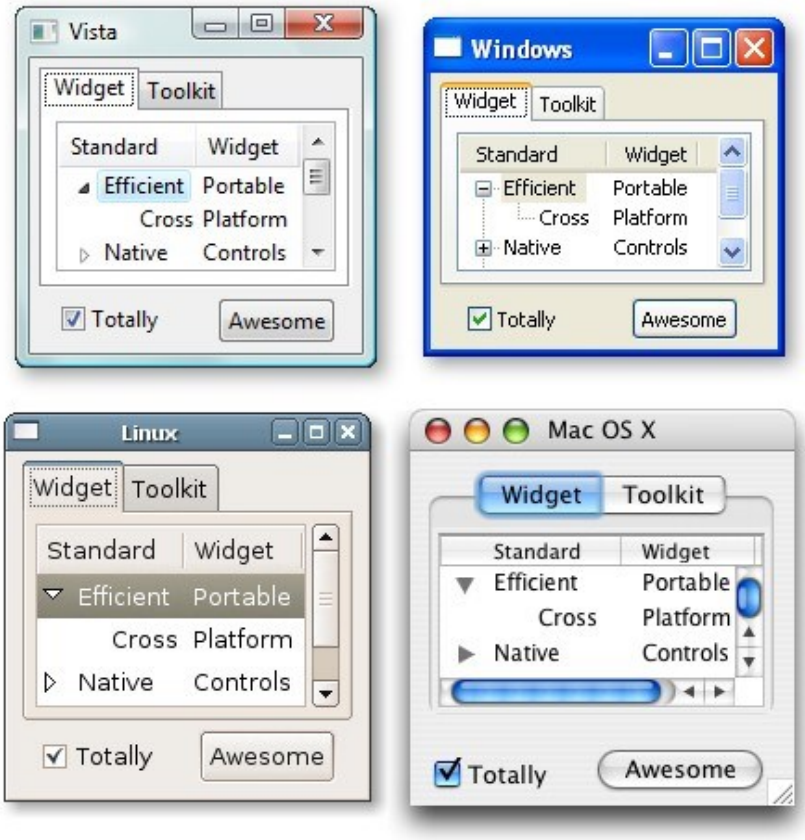


Obrázek č.1. Struktura wizardu [7]

JFace a SWT

JFace framework je definovaný Eclipsem jako nástroj, který poskytuje pomocné třídy pro vývoj uživatelské rozhraní při vytváření vlastního wizarda. JFace je nezávislý na jeho API, představuje systémové okno aplikace. Pracuje na úrovni systémového nástroje. Cílem JFace je usnadnit práci při vývoji, obsahuje jednoduché komponenty pro práci s frameworky určené pro wizardy, dialogy, fonty, obrázky atd. Obsahuje dva typy mechanismů pro práci s ním, prvním je zpracování příkazu uživatele v jeho uživatelském rozhraní, podle čeho určí, kdy má co nastat pomocí jednoduchých metod. Druhou funkcí je model grafického rozhraní [8].

JFace využívá pro grafické uživatelské rozhraní nástroj SWT, jež byl zmíněny výše v sekci 2.1 Eclipse, od verze 3 byl SWT nástroj použitý pro samotný vzhled Eclipse, ale i když byl později nahrazený jiným frameworkem, pro vývoj wizardu stále zůstává SWT. Tento nástroj slouží čistě je pro vykreslení GUI prvků, tyto prvky používá z nativní knihovny operačního systému, čímž se tyto aplikace, v našem případě wizard, stávají přenositelné mezi operačními systémy, ale jejich vzhled v odlišných operačních systémech je odlišný, jelikož každý systém používá svůj vlastní typ vzhledu, ukázka vzhledů v různých operačních systémech je zobrazena na Obrázek č.2, kde zleva doprava jsou v horním řádku vzhled Windows Vista a Windows XP, a ve spodním řádku je operační systém Linux a Mac OS X. [9]



Obrázek č.2. SWT v různých operačních systémech [10]

2.5 Java

„Základy Javy lze nalézt v projektu Oak, který vznikl ve firmě Sun na počátku devadesátých let pro řízení elektronických výrobků. V roce 1994 byl přenesen jako programovací jazyk do prostředí počítačů pod názvem Java (horká káva). Je to vyspělý multiplatformní programovací jazyk, obsahující všechny vlastnosti, které jsou vyžadovány v moderním programování, od modularity programu, řídících konstrukcí, přes silnou typovou kontrolu, multithreading, ošetření výjimek, správu paměti, i silnou podporu pro databáze, XML a síťové operace.

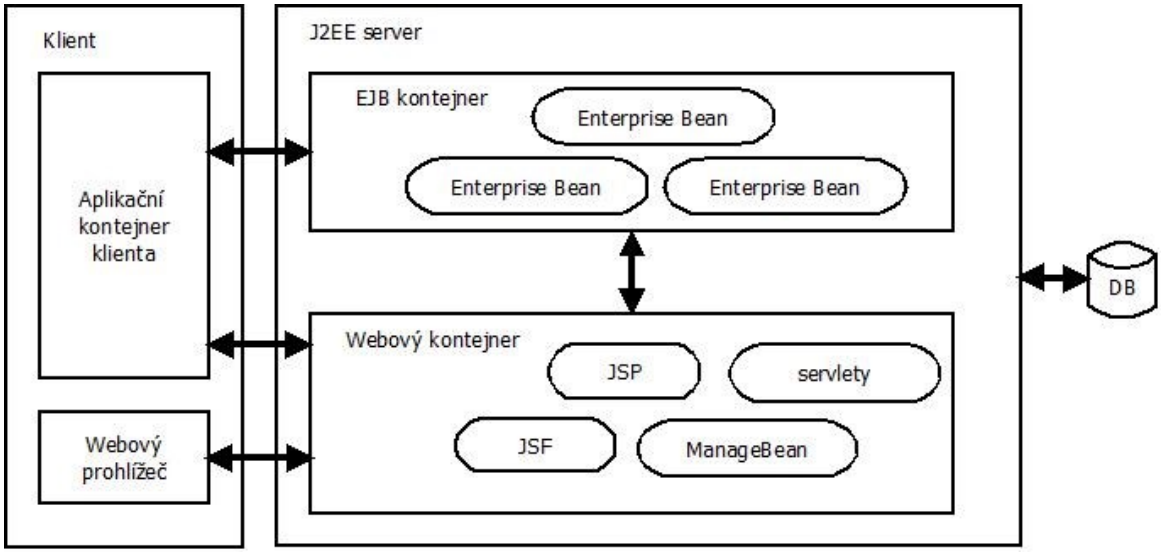
Zdrojový kód je při vývoji přeložen do spustitelného mezikódu (bytecode), který lze pak spouštět pomocí nainstalovaného runtime prostředí JVM (Java Virtual Machine), přímo na různých typech počítačů či technických zařízeních, od Linuxu, Unixu, Windows a další. Většina distribucí Linuxu již v sobě obsahují vývojové prostředky i runtime prostředí Javy, bez nutnosti nového překladu. Na rozdíl od jazyka C, který je rozšířen stejně nebo i více, by přenositelnost programu měla být dána ne pouze jen na úrovni zdrojového kódu, ale spustitelného programu, jak bylo řečeno výše.

Nižší rychlost, způsobená zpracováním v runtime prostředí může být urychlena s pomocí specializovaných překladačů na cílovém prostředí (Java just-in-time, JIT). I když základní vývojové prostředí obsahuje pouze řádkový překladač, existuje mnoho vývojových nástrojů a rozšíření dalších firem a autorů včetně IDE, i s podporou vývoje GUI aplikací.

Obsah Javy však nelze omezit jen na výčet jejích příkazů. Java je především silně objektová, což umožňuje v ní modelovat, vytvářet, používat a rozšiřovat rozsáhlé knihovny a systémy. Právě objektově je třeba myslet ne jen při psaní programu, ale již při návrhu a analýze. Pro tyto účely byl vytvořen modelovací jazyk UML, slouží k objektovému modelování a popisu konstrukcí reálného světa, převáděných do světa počítačů a informačních systémů." [11]

2.5.1 Java EE

Java EE je součástí platformy Java. Vychází z platformy Java SE, jenž nad ní vytváří součásti tvořící Java EE. Je určena pro provoz a vývoj v podnikových aplikacích. Hlavním cílem je zjednodušit vývoj aplikací tím, že poskytuje společný základ pro různé komponenty v Java EE ve vícevrstvých aplikacích. Komponenty jsou provozovány ve standardizovaném běhovém prostředí, kterých je k dispozici více od různých dodavatelů. Její vývoj vznikl v roce 1998 pod názvem JPE, o rok později již přijal název J2EE s verzí 1.2. Od verze 1.3 je rozvíjena pomocí skupiny JCP, kteří se starají o vývoj do dnes. V roce 2006 vypustila JCP verzi Java EE 5, kde se objevily velké změny v samotné definici kódu, od této verze přibyly anotace, které nahrazují a zastupují XML konfigurace, nové tagy pro psaní webových stránek v JSP i JSF. Poslední uvolněnou verzí je Java EE 7 z června 2013 [12].



Obrázek č.3. J2EE kontejner [12]

2.6 Webové stránky

Webové stránky jsou dokumenty, které jsou uchovávány na serveru, nebo na počítači. Tyto dokumenty obsahují jednoduché texty, pomocí nichž lze načíst a zobrazit třeba obrázky, videa, různé skripty a plno dalších věcí, jež slouží jako zdroj informací. Hlavním prvkem webových stránek jsou odkazy, slouží pro přechod mezi jednotlivými stránkami. Stránky můžou být buď statické, kde se jejich obsah nemění, anebo dynamické pro změnu obsahu stránek pomocí uživatele. Zobrazení webových stránek se pak provádí pomocí webových prohlížečů [13].

Pro jejich vytvoření existuje plno programovacích či skriptovacích jazyků, například můžou být tvořeny v Pythonu, asp, .NET, PHP, v Javě pomocí JSP a JSF a plno dalších. Jejich základ je především tvořen pomocí HTML, který obsahuje pár desítek definovaných tagů, pro tvorbu stránek.

Samotný zisk informací je založeny na http protokolu, jež vznikl pro předávaní dokumentů. Je založeny na základním principu HTTP požadavků. Funguje to tak, že se prohlížeč nejprve připojí k potřebnému webovému serveru, následně zašle serveru svůj požadavek, který server zpracuje a odešle prohlížeči přes vytvořené připojení HTTP odezvu a zruší toto připojení. Každý požadavek musí obsahovat 3 mezerou oddělené pole, tím je metoda požadavku, požadovaná adresa URI a verze HTTP. Existují dvě nejčastěji používané metody požadavku, jsou to požadavky GET a POST [14].

2.6.1 GET požadavek

Tento požadavek slouží pro získání určité informace o nějakém objektu ze serveru. Posílané informace se kódují pomocí standardu MIME, tyto data se připojují k požadované URI adrese za znak otazníku. Data se posílají pomocí dvojce klíče a hodnoty, tyto data lze řetězit pomocí ampersandu, samotná délka URI je omezená [14].

2.6.2 POST požadavek

Požadavek POST plní stejnou funkci jako GET pro získávání dat ze serveru nebo posílaní na něho. Oproti GET požadavku se nevkládá požadavek do URI adresy, ale vkládá se jako speciální zpráva do obsahu požadavku. Využívá opět standardu MIME, tudíž i stejný formát jako GET. Velký rozdíl je v množství přenesených dat, protože URI adresa je omezena velikostí, kdežto POST požadavek nemá žádné omezení. Jelikož POST požadavek je posílán v těle požadavku, tak není vidět jeho tělo oproti GET, který je obsažený v URI adrese [14].

2.7 Java a webové stránky

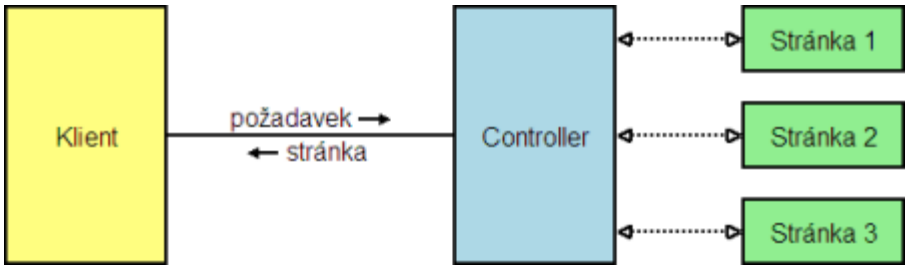
2.7.1 JSP a servlety

Java servlety jsou nízko úrovňové nástroje pro obsluhu protokolu HTTP na serverové straně, kde fungují na způsob požadavek-odpověď. Servlety jsou v javě třídy, které jsou potomkem třídy HttpServletu. Každý servlet obsahuje metody doGet a doPost s parametry požadavku a odpovědi pro zpracování HTTP požadavku. Jejich myšlenka byla vytvářet webové stránky podle HTTP požadavku, ale tento způsob není vhodný. Dnes slouží v JSP stránkách jako kombinace statického HTML s dynamickým generovaným obsahem [15].

JSP stránky byly vytvořeny jako nadstavba servletů. Hlavním důvodem vzniku této technologie je kvůli nízko úrovňové obsluhy HTTP požadavků a nepohodlnému psaní HTML značek do Java metod jako obyčejný string, a díky tomu není možnost kontrolovat správnost syntaxe napsaného HTML kódu, kde u rozsáhlejších kódů je příliš problematické ohlídat správnou syntaxi [16].

První JSP stránky vznikly jako servlety naruby, kde JSP stránka představovala prostý text, HTML text, do kterého se vkládal Java kód pomocí speciálních značek, tyto stránky pak byly před použitím pře generovány na servlet. V dalších verzích byla platforma Java EE rozšířena o knihovnu tagů JSTL, které pomáhají rozšířit JSP soubory o nové vlastnosti jako jsou jednoduché podmínky, cykly, vkládaní parametrů do požadavků pro servlety, formátovací funkce. Samotná knihovna JSTL obsahovala i jazyk EL, který byl později přesunutý do samotné specifikace JSP. EL jazyk rozšiřuje vlastnosti JSTL tagů, pro snadný přístupu k datům, přes uložené atributy, požadavky stránek, nebo přes session.

JSP stránky společně se servlety vytvářejí model MVC, kde pomocí jednoho řídicího servletu lze lépe zajistit bezpečnost a logování, a tím se stávají jednotlivé JSP stránky méně závislé na ostatních stránkách, tato struktura je zobrazena na Obrázek č.4. [17].



Obrázek č.4. MVC model JSP stránek se servletem [17]

2.7.2 JSF

Technologie JSF byla uvolněna v roce 2004 jako náhrada dřívějších JSP stránek. Jádrem byly právě JSP stránky se servlety, které byly rozšířeny o nové knihovny tagů. Hlavní příčinou,

proč byla vyvinuta tato technologie, bylo zjednodušení vývoje webových stránek, obsluhy požadavků a jednoduchost použití. JSF stránky jsou, stejně jako JSP, založeny na modelu MVC. Technologie využívá java beany pro zpracování událostí a držení stavu aplikace, získání dat lze provádět až v samotném kódu JSF stránek pomocí EL, tudíž se už nemusí provádět všechno přes požadavky. Ve verzi 2 vydané v roce 2009 se výrazně změnily možnosti vývoje JSF stránek, získaly svůj vlastní šablonovací framework nazvaný „facelets“. Od této verze můžou java beany obsahovat anotace pro zjednodušení vývoje, odpadá tím obtížné zapisování údajů do XML souborů. Komunikace s java beanami se provádí okamžitě, a to i pro uložení právě vložených hodnot. Poslední vydaná verze 2.2 z dubna 2013 obsahuje oproti verzi 2 jen pár drobných úprav. [18]

2.7.3 JPA

JPA je součástí specifikace EJB 3.0 z J2EE 5, cílem bylo zjednodušit načítání dat z databáze, jedná se o mapování Java objektů z relační databáze, tudíž objektově-relační mapování. Pomocí JPA lze nastavit obyčejné POJO třídy na JPA entity, jež představuje persistentní data uložená v relační databázi, kde určitý objekt JPA třídy představuje konkrétní instanci z databáze.

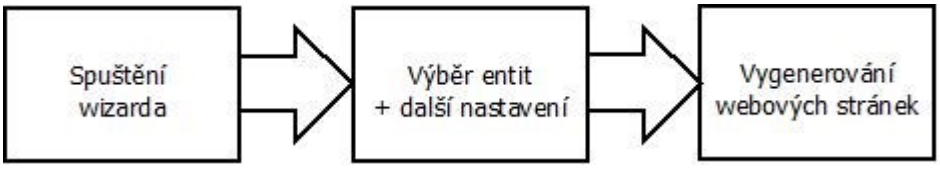
JPA entity musejí splňovat některé požadavky, aby mohla být prohlášeny za entity. Musí obsahovat bezparametricky veřejný konstruktor, třída a metody nesmí být deklarovány jako final, může dědit i z obyčejných POJO tříd. Atributy musí být deklarovány jako private nebo protected, a pro přístup k ním lze provést pouze přes get/set metody.

JPA obsluhuje objekty entit pomocí třídy EntityManager, který nabízí 4 základní metody pro práci s objekty na uložení, smazání, úpravu a vyhledání instance v databázi. Pro pokročilejší dotazy lze využít Java Persistence Query Language, který je podobný SQL [19].

3 Analýza a návrh aplikace

V této části bude nejprve jednoduše popsaný celý projekt, a to tak, aby mu rozuměl obyčejný zákazník. Pak se podíváme na specifikaci požadavků, kde si pomůžeme několika diagramy. Pak bude podrobně rozepsaná analýza a bude provedený návrh projektu celého projektu z pohledu návrháře, kde bude podrobnější rozebrání jednotlivých problému s potřebnými UML diagramy pro návrh aplikace, čímž dostaneme model návrhu, který upřesní a přenese model analýzy do stavu, aby byla následná implementace co nejjednodušší. Určitě se nesmí ani zapomenout na vytvoření prototypu uživatelského rozhraní, ať víme, jak následně GUI vytvořit.

Cílem celé práce bude generovat webové stránky typu JSP nebo JSF podle výběru uživatele, a to ze zvolených entit. Výsledná aplikace bude představovat plugin pro Eclipse, který bude v Eclipse zobrazený pomocí wizarda a jeho několika stránkami. Až si uživatel nainstaluje tento plugin, tak bude mít v Eclipse tlačítko na tohoto wizarda, pro jeho spuštění musí být nutné si zvolit projekt, s kterým chce uživatel pracovat, wizard mu zobrazí seznam všech existujících entitních tříd, jež jsou ve vybraném projektu obsaženy. Uživatel si následně vybere jen ty entity, z kterých chce vytvořit webové stránky. Uživatel bude moci si zvolit i možnosti, jaké typy operaci budou moci stránky vykonávat, tím je myšleno přidávání, odebírání a mazání položek ve vybrané entitě. Také musí existovat volba výběru úložiště vygenerovaných souborů, a to jak pro webové stránky, tak i pro jejích zdrojové java soubory. Výsledkem celého wizardu budou vygenerované funkční webové stránky, podle nastavení, které uživatel předal wizadru.



Obrázek č.5. Znázornění kroků pluginu

3.1 Vize

Proč?

Plugin bude sloužit k vygenerování kompletních funkčních webových stránek z vloženého entitního modelu.

Co?

Vyberou se entity z vybraného projektu.

Jak?

Vygenerují se kompletní webové stránky, které odpovídají vloženým entitám.

Kde?

Práci s aplikací lze provádět na každém uživatelském počítači, který obsahuje Eclipse J2EE a vytvořený plugin.

Kdo?

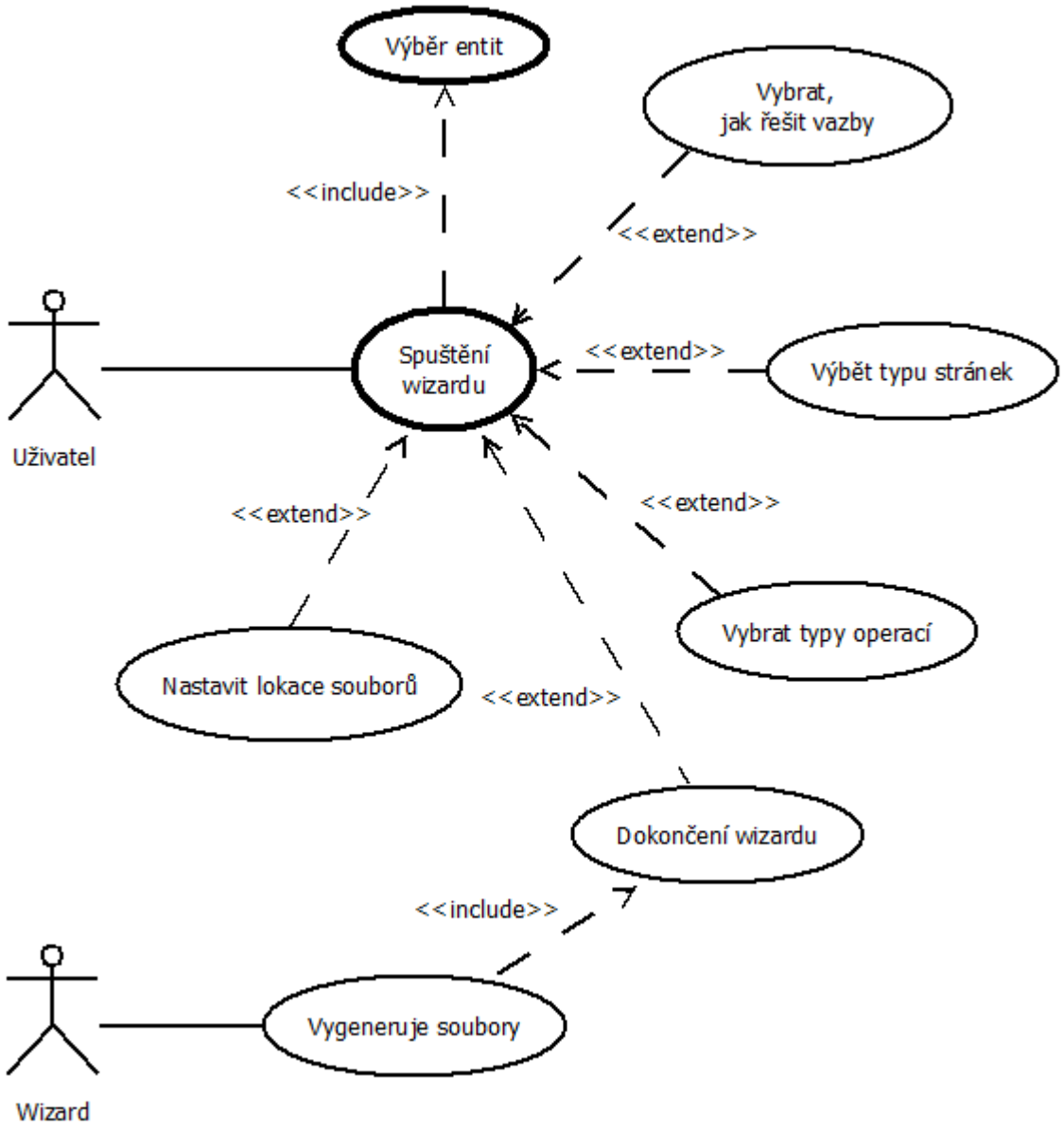
Aplikaci může používat kdokoliv, kdo ovládá základní znalosti jazyka Java.

3.2 Specifikace požadavků

3.2.1 Funkční požadavky

Prvním důležitým krokem v pluginu je spuštění wizarda, který musí obsahovat formulář pro důležitá data. Nejdůležitějším krokem je výběr existujících entit z vybraného projektu s minimálním počtem 1. Protože entity, které přestavují databázové entity, jež běžně mezi sebou můžou mít libovolné vazby, tak nastává problém, když se z vazby vybere pouze jedna entita, v tom případě musí existovat možnost výběru, co s tou druhou entitou, tady se nabízí dvě možnosti, a to jestli druhou entitu načíst nebo celou vazbu zahodit. Uživatel by měl mít možnost definovat, jaký typ stránek se má vygenerovat, buď JSP nebo JSF stránky, a jaké funkce májí stránky obsahovat, tím je myšleno vytváření, úprava a mazání objektů z entit. Základní vlastnost, kterou by stránky měli mít, je prohlížení objektů vybrané entity. Uživatel by měl mít také možnost si určit, kde se mají uložit vygenerované webové stránky a jejich javovské zdrojové soubory. Všechny tyto požadavky jsou popsány v Use Case diagramu na Obrázek č.6.

3.2.2 Use Case diagram



Obrázek č.6. Use Case diagram

Spuštění wizardu:

Aktéři: uživatel

Typický průběh: Wizard se spustí a otevře se stránka s entitami

Alternativní průběh: Wizard se spustí s varovnou chybou

Prekondice: Musí být vybraný projekt a vybraný projekt musí obsahovat alespoň jednu entitu

Postkondice: Otevře se wizard s první stránkou s výběrem entit

Nastavit lokace souborů:

Aktéři: uživatel
Typický průběh: Lokace souboru má vložené správné oddělovače
Alternativní průběh: Jsou vloženy špatné oddělovače
Prekondice: Musí být vybrané entity

Dokončení wizarda:

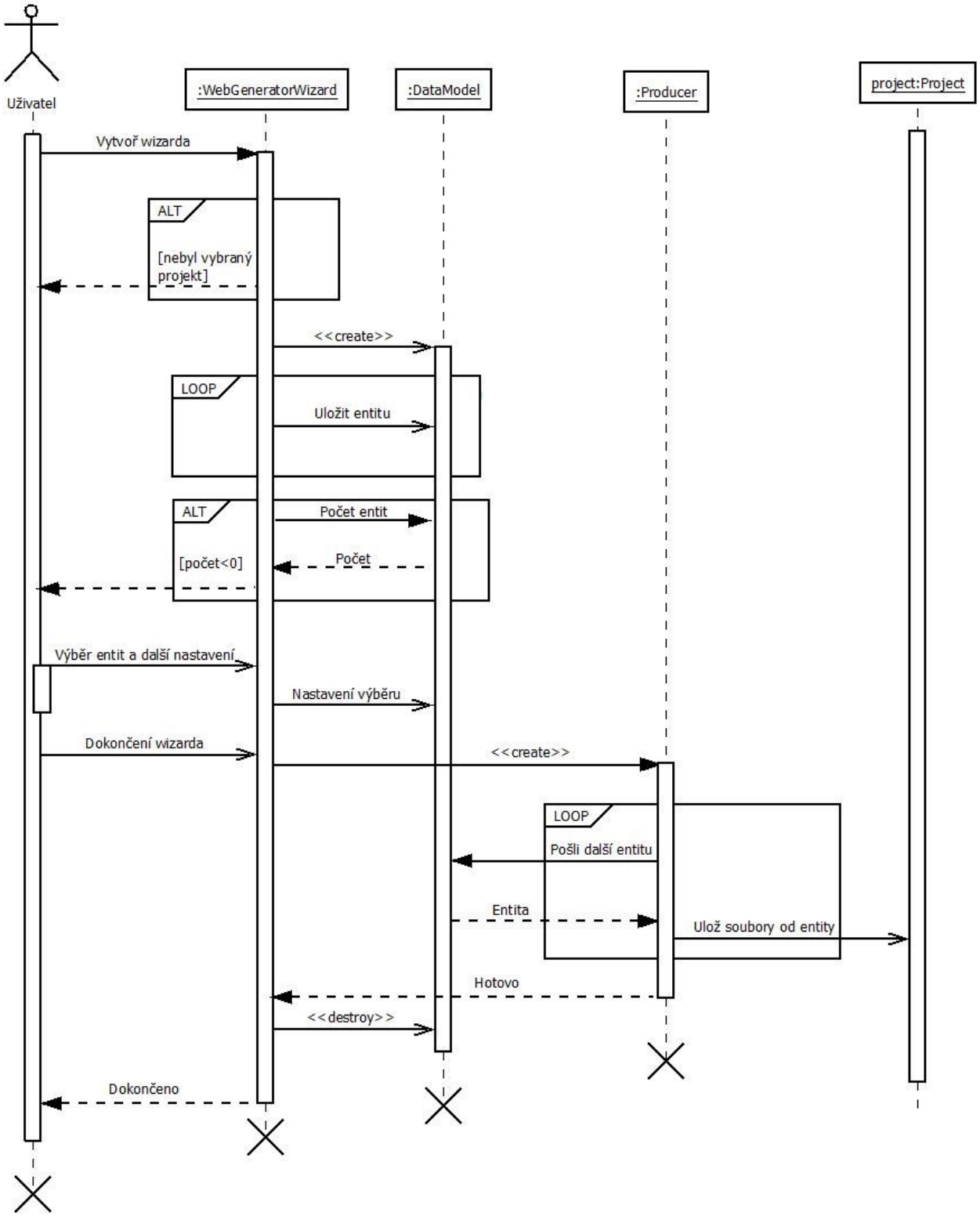
Aktéři: uživatel, wizard
Typický průběh: Wizard se úspěšně dokončí
Prekondice: Musí být vybrané entity
Postkondice: Vygenerují se požadované stránky

Vygenerování souboru:

Aktéři: wizard
Typický průběh: Úspěšně vygeneruje stránky
Alternativní průběh: Nevygeneruje stránky (chybné entity nebo lokace)
Prekondice: Musí být vloženy všechny potřebné informace
Postkondice: Vygenerují se webové stránky s

3.2.3 Sekvenční diagram

Průběh chování uživatele, jeho zprávy a nastavení, a chování wizarda zachycuje sekvenční diagram na Obrázek č.7, zobrazuje posloupnost událostí v daném časovém okamžiku práce uživatele s wizardem od začátku až po dokončení vygenerování webových stránek. Na začátku vydá uživatel zprávu na spuštění wizarda, ten zkontroluje, jestli byl vybraný projekt, s kterým má dále pracovat. Pokud nebyl vybraný projekt, tak dále nepokračuje. V opačném případě vytvoří datový model a vyhledá všechny entity v projektu, které uloží do vytvořeného datového modelu, pokud nenajde žádnou entitu, tak wizard nebude dále pokračovat a skončí. Když má vybraný projekt alespoň1 entitu, tak uživatel vybere nějaké z těchto entit a nastaví si potřebné hodnoty pro vygenerování stránek, tyto hodnoty si wizard uloží do datového modelu. Až si uživatel nastaví všechny požadované věci, tak dokončí wizarda, aplikace zpracuje vstupní data a spustí generátor webových stránek. Generátor projde všechny uživatelem vybrané entity a z každé entity vygeneruje potřebné soubory pro webové stránky a uloží je do projektu. Po vygenerování stránek ze všech uživatelem vybraných entit wizard skončí.



Obrázek č.7. Sekvenční diagram

3.3 Analýza a návrh řešení

V této části bude podrobně rozebrána problematika vnitřní funkcionality pluginu, oproti předchozím bodům, kde se řešila funkcionalita hlavně z pohledu uživatele. V prvním kroku se podíváme a zaměříme nejprve na vstupní data, především na možné typy a vlastnosti entit, jejich property a jejich důležitost při pozdějším vygenerování samotných webových stránek. Pak se zaměříme na základní funkce uživatele ze specifikace požadavků. Jednotlivé akce budou pečlivě prozkoumány a odhaleny jejich možné problémy, které můžou nastat. U každé akce se podíváme trošku hlouběji a vytvoříme vhodné řešení pro následný vývoj pluginu. Nakonec budou řešeny všechny vygenerované soubory, jejich výstupní formáty a možnosti, které jsou závislé na vstupních informacích uživatele.

3.3.1 Vstupní entity

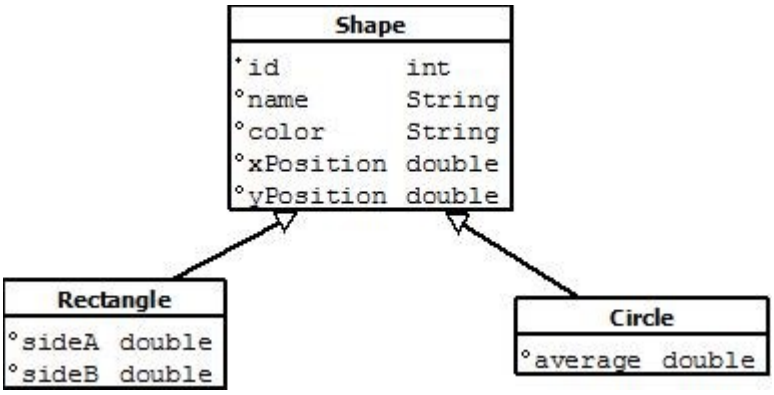
Samotné entity jsou obyčejné java třídy, které nám reprezentují objektově orientovaný pohled na libovolnou entitou uloženou obvykle v relační databázi. Tyto soubory jsou uloženy na serverové části aplikace, tudíž v EJB kontejneru. To znamená, že každá instance dané entitní třídy představuje data jednoho objektu dané tabulky z databáze, a to pomocí objektově relačního mapování, které provádí JPA.

Rozpoznání entit

S příchodem Javy EE 5 byly vydány nové EBJ verze 3.0, které přinesly celou řadu novinek. Týkalo se to i entitních tříd, jelikož od této verze se už nemusí zapisovat každá entita a její potřebné vlastnosti do XML souborů, ale stačí použít definovanou anotaci z balíčku javax.persistence.*. Takže každá entita definovaná tímto novým způsobem nese anotaci `@Entity`. Protože dnes už je tento způsob vytváření entity běžný, tak starou implementaci entit budu dále ignorovat a budu entity vyhledávat a rozeznávat od ostatních java souborů právě přes tuto anotaci [20].

Předci entit

Každá taková entita má možnost dědit maximálně z jednoho svého předka a to i z ne-entitní třídy. Na Obrázek č.8 je ukázka, k čemu se dědění používá, tudíž nemusí se psát ke každému potomkovi všechny jejich stejné atributy a metody, tyto vlastnosti zdědí z předka. V mém případě bude důležité tyto informace z předka zjistit a uložit je, princip jak si tohle pak dále řeší databáze, nás už nezajímá.



Obrázek č.8. Dědění v databázi

Atributy

Všechny správné atributy entity nesmí být deklarovány jako static a měly by být private nebo protected. Každý z těchto atributů musí mít podle potřeby vygenerované get/set metody, aby se k nim dalo přistupovat. Povolené datové typy jsou primitivní java datové typy, String, typy pro práci s daty, byte pole, pole znaků, BigIntegery a BigDecinal, enumerační typy, což jsou předem nadefinované textové řetězce pro danou property, dále ještě může být datovým typem jiná entita nebo její kolekce.

Ostatní anotace

V entitě můžou existovat i další anotace, které jsou přiřazeny právě atributům, jednou z nejhlavnějších je anotace `@Id`, která určuje jednoznačný identifikátor entity. V aplikaci tato anotace znamená, že musí být tato hodnota vložena a po vložení nesmí být dále upravována, tuto hodnotu musí mít každá instance dané entity odlišnou, aby se dala každá instance přes tuto hodnotu jasně identifikovat, pro plugin je tato hodnota velmi důležitá, protože pro zobrazení instance, její úpravu nebo smazání je nutné tuto instanci identifikovat právě přes tuto Id hodnotu. Pro automatické vygenerování Id hodnoty existuje anotace `@GeneratedValue`, tudíž v tomto případě by nemělo být povolené ani vkládání při vytváření nového objektů. Tato anotace obsahuje různé parametry, odkud se mají hodnoty generovat, popřípadě jak se mají generovat, ale tohle je opět část, kterou si řeší server sám.

Dalšími hodně důležitými anotacemi jsou anotace definující vazbu. Jak je známo z databází, můžou existovat 4 typy vazeb 1:1, 1:N, N:1, M:N. Tyto vazby je důležité rozlišovat kvůli následnému typu zobrazení a výběru položek při vytváření či změně dat. Anotace každé vazby by měla obsahovat parametr o mapování, tato informace je pro mě důležitá, protože určuje, ve které entitě ve vazbě se vazba nastavuje. Volitelným parametrem anotace vazeb je způsob načítání vazební entity, ve výchozím stavu je nastaveno „líné“ načítání, což mi říká, že se nenačtou informace o vazební entitě předem, tudíž tyto údaje nemůžu ani zobrazovat, v opačném případě musí být nastavený parametr Fetch na hodnotu „EAGER“, čímž se načtou všechny informace o druhé entitě ve vazbě, takže můžu tyto informace zobrazit.

I když Java obsahuje datové typy týkající se data a času, tak i pro tento typ byly vytvořeny anotace, aby se mapovaly na správný datový formát. Přes anotaci `@Temporal` se určí, že se bude jednat o typ s datem nebo časem. Entity dokážou přijmout pouze datový typ java.util.Date a java.util.Calendar. Tato anotace má jeden parametr, který nese typ data nebo času, jak je ukázáno v Příklad č.1. Jelikož existuje mnoho různých formátu dat a časů, tak je při implementaci nutné zvolit nějaký jeden určitý formát, ve kterém se pak budou muset všechny data a časy vkládat a zobrazovat. To znamená, že musí být vytvořený převodník mezi vloženými daty a datovým typem Date nebo Calendar. Vkládání dat se dá řešit dvěma způsoby, zaprvé můžeme rozsekat vstupní pole na více vstupních polí, podle počtu potřebných údajů, anebo nechat jen jedno pole, s tím že uživatel bude nucen vkládat datum ve správném formátu.

```
@Temporal(DATE)
@Temporal(TIME)
@Temporal(TIMESTAMP)
```

Příklad č.1 Ukázka práce s anotací pro data a časy

Pomocí anotací nemusí být property nadeklarovaná pomocí proměnné, ale lze přidělit anotaci i get metodám, tento princip se nazývá nepřímým přístupem k property. Pokud je metoda správně anotovaná, tak zvenčí vidíme tuto metodu jako property.

V Příklad č.2 je ukázka nepřímého volání property, kde `fname` a `lname` nemají get/set metody, ale jelikož mají v databázi své zastoupení, tak jsou jejich data inicializována, ale není k nim přímý přístup pro zisk nebo změnu. Na řadu přichází metoda `getName`, která představuje nepřímou property, v tomto případě nese anotaci `@Column`, ale může obsahovat i jiné anotace, třeba `@Id`, atp. Tato metoda, která se chová a volá se přes EL jako property `name`, poskytuje na výstupu nepřístupné property `fname` a `lname`, vrací je jako jeden řetězec.

```
Private String fname;
Private String lname;

@Column
Public String getName() {
    return fname+lname;
}
```

Příklad č.2 Nepřímé volání property

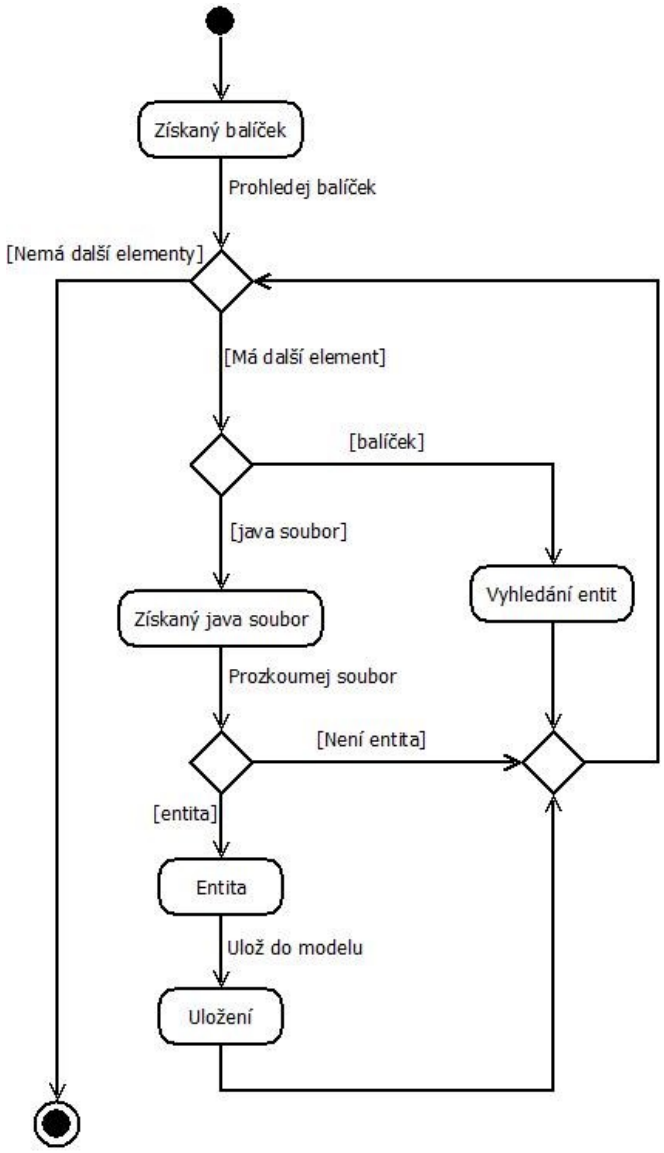
Ostatní anotace, které nabízí Java EE od verze 5, jsou buď zanedbatelné, protože nijak neovlivňují výstupní kód či formulář, anebo jsou vytvořeny pro serverovou část aplikace, pro nějakou automatizaci na serveru, či ulehčení práce uživateli. A proto tyto anotace nebudou vůbec řešeny [21].

3.3.2 Akce uživatele

Na začátku, musí mít uživatel spuštěný Eclipse s nainstalovaným pluginem. Nejprve vybere projekt, a spustí plugin. Po této akci by se mu měl spustit wizard, který po něm bude chtít potřebné údaje.

Vyhledání entit

Při spuštění wizarda, se musí zkontrolovat existence projektu, protože když nebude vybraný projekt, tak wizard nemůže vybírat a ukládat žádné soubory, když nemá místo, kde by je uložil. Pokud bude tato podmínka splněna, tak následuje vyhledávání všech existujících entit v projektu. Princip je jednoduchý, stačí projít v projektu všechny balíčky a v nich všechny java soubory, entity jsou pak ty třídy, které obsahují anotaci `@Entity`. Celý postup je zobrazený v diagramu na Obrázek č.9.



Obrázek č.9. Hledání entit v balíčku

Nastavení údajů uživatelem

Po vyhledání všech entit musí zkontrolovat wizard, jestli je nalezena alespoň jedna entita, ať může uživatel vybrat nějaké entity, jelikož výběr entit je základem celého pluginu, bez entit by nešly vygenerovat žádné webové stránky ani související java soubory. Uživatel si vybere entity a vyplní ve wizardu potřebné informace. Pro dokončení nesmí wizard pustit uživatele dále ke generování stránek, pokud nevybral alespoň jednu entitu, to je jediný nutný požadavek, jenž musí uživatele splnit. Všechny vložené uživatelovy údaje wizard uloží do datového modelu.

3.3.3 Kontrola správnosti entit a vazeb

V této sekci se již začínáme zabývat vnitřní problematikou, tudíž samotným generováním webových stránek. Hlavní roli zde budou hrát opět především entit. Cílem je z vybraných entit vytvořit strukturu shluků entit propojené vazbami. Postupně projdeme všechny vybrané entity uživatelem a zkontrolujeme, jestli dědí z nějakého předka, pokud dědí, tak daná entita získá všechny property tohoto předka.

Ověření property

Když máme všechny property dané entity, tak je nutné je ověřit, zda jsou platné a potřebné pro následné generování java souborů a webových stránek. U každé property se musí ověřit několik věcí, jsou to get/set metody, nepřímé volání property, identifikátory a vazby. Díky tomu dostaneme platnost property, popřípadě pak celé entity.

U každé property je nutné ověřit, jestli k nim v entitě existují jejích get/set metoda. Jak bylo výše uvedeno, uživatel má možnost výběru operace na vygenerovaných webových stránkách, tím je myšleno vkládání nových objektů do databáze, jejich úprava a smazání, prohlížení objektů dané entity z databáze se bude brát jako povinná akce. Na těchto uživatelem vybraných operacích při průzkumu property záleží, protože akce vkládaní a úpravy vyžadují mít set metodu, v opačném případě, když nebude ani jedna z těchto dvou akcí vybraná, tak není nutná set metoda. A tohle je důležité ověřit, neboť například může mít uživatel vytvořené entity pouze s get metodami k atributům, a při generování nemusí vybrat žádnou operaci, popřípadě jen mazání instancí z databáze, což znamená, že stačí mít v entitách pouze get metody, a to by měl uživatel splněné. Ale kdyby si uživatel vybral, že chce navíc vytvářet instance nebo je upravovat, tak bez set metod to nepůjde. Z toho vyplývá, že když není vybrané vkládání ani úprava, tak stačí vyhledat jen get metodu k property, v jiném případě musí být nalezena get i set metoda, aby byla daná property platná.

Platnost entity

Tím, že uživatel vybere entitu, není zaručené, že se z ní stránky musí vždycky vygenerovat. Každou entitu je nutné před samotným generováním stránek zkontrolovat. Způsob ověření je prostý, nesmí jednoduše existovat entita bez jediné platné property, entita sice může obsahovat plno property, ale pokud nemají potřebné get/set metody, tak to jako kdyby entita neexistovala. Z toho plyne, že každá entita musí mít alespoň jednu platnou property.

Anotace vazeb

Jednou z nejdůležitějších anotací property, jsou anotace vazeb, definují nám vazbu mezi dvěma entitami. Výše bylo zmíněno, že existují 4 druhy vazeb, tudíž i 4 různé anotace, které mapují jeden z těchto druhů. Na tento problémem se váže uživatelovo nastavení povolení automatického přidávání referencí, protože kdyby tuto volbu uživatel nenastavil a vybral by entitu s vazbou na druhou entitu, kterou by nepřidal dále pro generování stránek, tak se tato vazba nemůže uložit jako platná vazba. Jiným případem je, když vybere obě entity, kde se musí

obě entity zkontrolovat, jestli jsou platné. V případě, že jsou obě platné, tak se tato vazba stává také platnou.

Při zvolení automatického přidávaní referencí se u vazby musí vždycky zkontrolovat platnost druhé entity ve vazbě. Pokud tato entita byla již úspěšně zkontrolována, tak se vazba stává platnou. Ale když entita nebyla ještě zkontrolována, tak se entita vyhledá v projektu, a to buď v seznamu uživatelem vybraných entit, anebo v nevybraných entitách. Až je tato druhá entita nalezena, tak se musí vyřešit problém pro ověření této entity, pokud bude entita platná, tak se i vazba stává platnou.

Nepřímé volání

Nepřímým voláním je myšleno načítání property přes metodu, která není definovaná jako property, tak jak bylo ukázáno v Příklad č.2. To znamená, že nestačí prohledat pouze property, ale musí se projít i anotace metod, musí to být metoda typu get. Jestliže bude nalezena nějaká taková get metoda s anotací, tak se k této metodě podle potřeby, záleží na uživatelem vybraných operací, musí vyhledat i druhá metoda set, aby byla property platná.

Ostatní anotace

Kromě anotací vazeb jsou v entitách i další možné anotace. Asi nejdůležitější anotací je `@Id`, tuto anotaci musí obsahovat každá entita přesně jednou, tudíž tato property musí být vždycky platná, jinak se stává celá entita neplatnou, protože nám určuje jednoznačný identifikátor na daný objekt entity. Na tuto anotaci se může vázat další anotace, pro automatické číslování. Tyto dvě anotace, je nutné najít a speciálně uložit, aby bylo možné se na stránkách odkazovat na daný objekt, pro jeho práci s ním. Tuto anotaci lze uvádět, což znamená nutnost hledat, u property i u metod, pro nepřímé volání property.

Existuje jeden speciální typ property, který se nazývá enum. Pro deklaraci toho typu v entitě slouží anotace `@enum`. Tento typ je výčtový typ, v překladu to znamená, že obsahuje seznam řetězců, které slouží jako omezení pro jinou proměnou, která smí obsahovat jen jeden řetězec z daného výčtu hodnot z enumu. Hlavním cílem při nalezení této anotace, je potřeba najít tento výčtový typ v projektu, buď přímo v entitě, kde se anotace enumu vyskytuje, popřípadě jako samostatný soubor enum, anebo může být deklarován uvnitř jiné třídy.

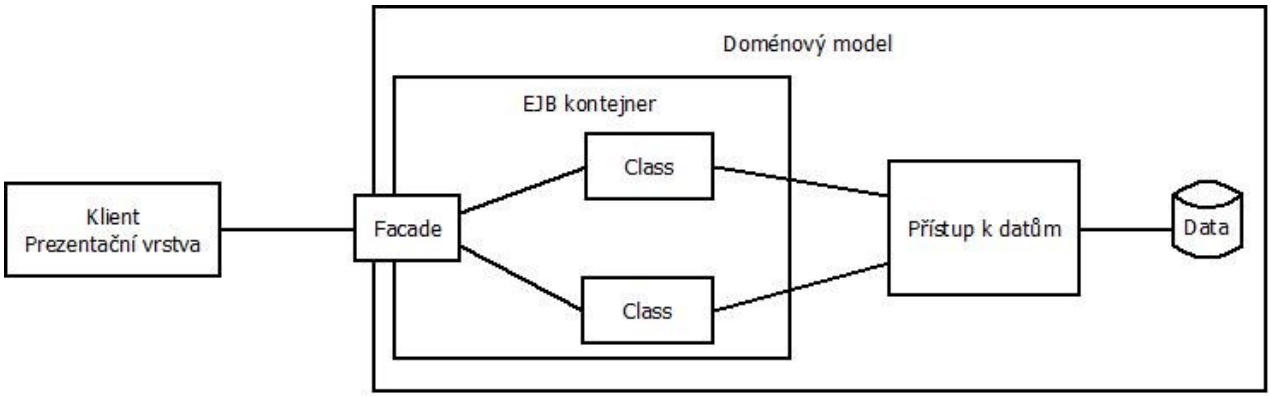
Pro anotaci pro práci s daty, je nutné ošetřit převod vstupního data do správného datového typu, důležitou roli zde hraje vkládaný formát. Existuje plno formátu data a jejich modifikací, proto si zvolím formát typu yyyy-MM-dd HH:mm:ss. Ostatní anotace jsou už méně potřebné, popřípadě jsou určeny pro serverovou část, což si řeší samotný server.

3.3.4 Generování souborů

Jakmile jsou zkontrolovány všechny uživatelem vybrané entity a je vytvořený seznam entit splňující nutné podmínky, aby z nich mohly být generovány stránky, tak začne generování všech

potřebných souborů nutných pro chod webových stránek ve vybraném typu. Jedinou podmínkou je, aby tento seznam entit, z kterých se mají vytvořit stránky, obsahoval minimálně jednu entitu.

Vygenerované webové stránky v JSP a JSF se budou vytvářet podle návrhového vzoru nazvaném "fasáda" (přeloženo z anglického názvu "facade"), tento vzor dělí vygenerované stránky na dvě části. První z nich je EJB kontejner, jež je součástí doménového modelu, který přímo přistupuje k datům přes JPA entity. Druhá část jsou soubory prezentační vrstvy, pokud chce prezentační vrstva získat data, musí se k nim nějak dostat, a to je právě skrz návrhový vzor fasáda, který obaluje celou doménovou část. Dá se říct, že vytváří jediné vstupní a výstupní místo v doméně, což je v podstatě API rozhraní domény pro přístup z vnějšku.

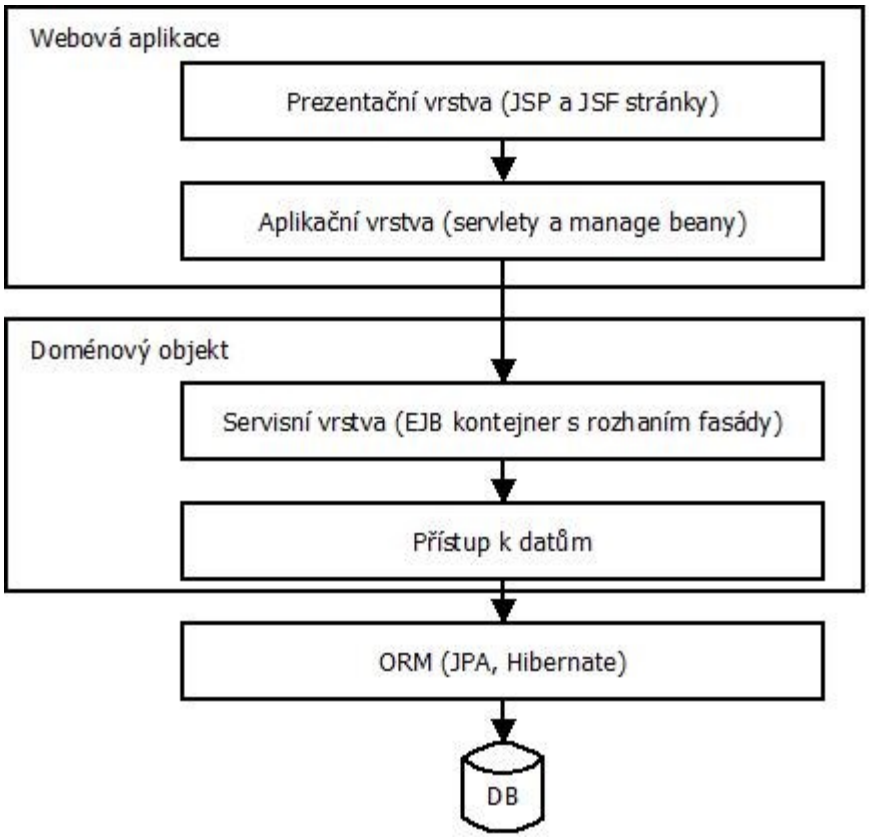


Obrázek č.10. Návrhový vzor „Facade“ pro JSP/JSF stránky

Typy souborů a jejich struktura generování

Výsledný plugin nebude generovat pouze jeden typ souboru, výsledkem bude několik typů souborů. Protože ve výsledné aplikaci má být na výběr ze dvou typů webových stránek, jež budou generovat dva rozdílné typy kódů, tak musím vytvořit vhodnou strukturu, a to tak, aby měl plugin možnost být v budoucnu dále rozšířen o nové typy webových stránek. V mém případě dostanu strukturu pro 2 typy stránek.

Vygenerované webové stránky budou obsahovat několik vrstev, první je prezentační vrstva se samotnými webovými soubory, která bude podle zvoleného typu stránek buď JSP nebo JSF. S touto vrstvou je svázaný kontrolér, který leží v aplikační vrstvě a komunikuje s výše zmíněným rozhraním z návrhového vzoru fasáda, který je součástí EJB kontejneru v doménovém objektu. V doménovém objektu proběhne komunikace mezi EJB kontejnerem a vrstvou pro datový přístup, s pomocí knihovny zajišťující mapování dat na objekty, celá struktura je na Obrázek č.11. Plugin by měl vygenerovat soubory od prezentační vrstvy do vrstvy pro přístup k datům.

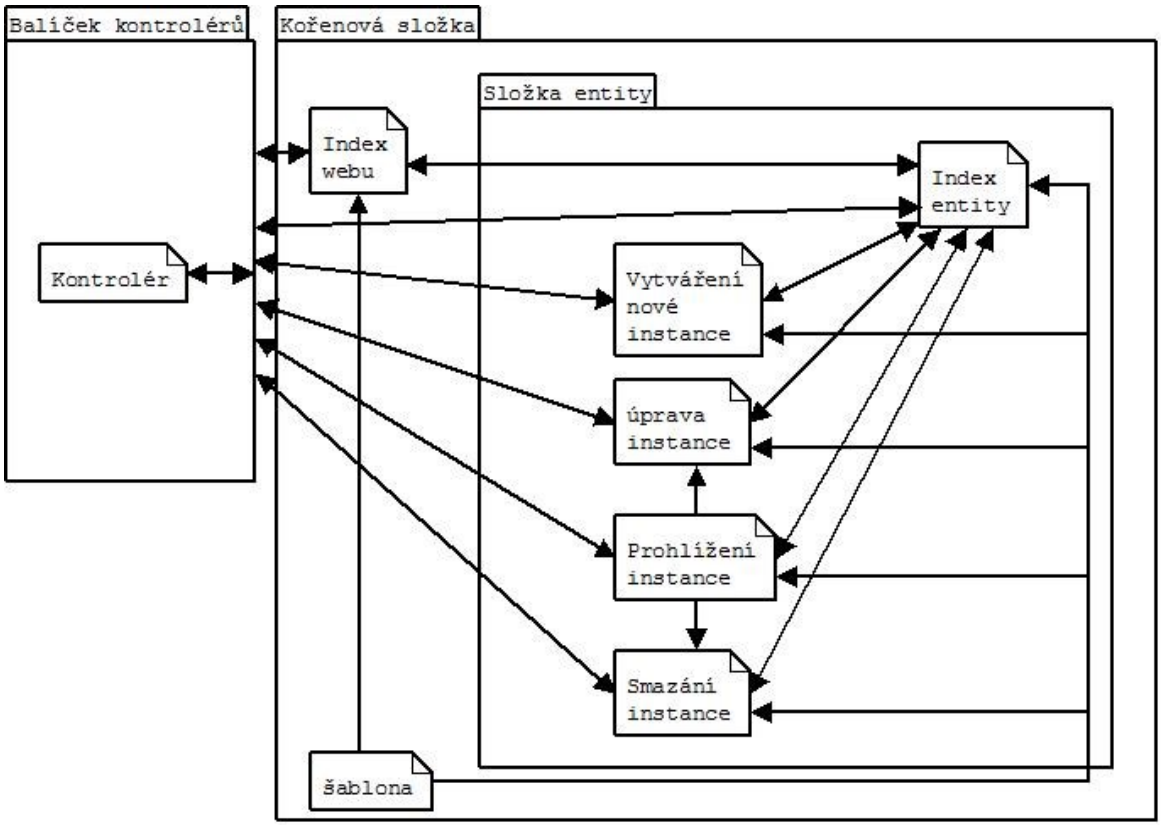


Obrázek č.11. Struktura webových stránek

Vrstvy v doménovém objektu mají předem jasnou strukturu pro všechny vybrané entity, kde soubory pro přístup k datům budou obsahovat základní operace s daty (vkládaní, úprava, mazání, výběr všech instancí, výběr jedné instance entity), pro práci s datovým zdrojem se využije objektově relační mapování. Servisní vrstva bude obsahovat pouhé mapování metod na vrstvu pro přístup s daty, jejím základním cílem je obalit doménový objekt rozhraním podle vzoru fasáda (Obrázek č.10). Z toho plyne, že každá z obou těchto vrstev bude potřebovat pro každou entitu téměř stejné třídy, které se budou lišit jen v určitých pojmenováních.

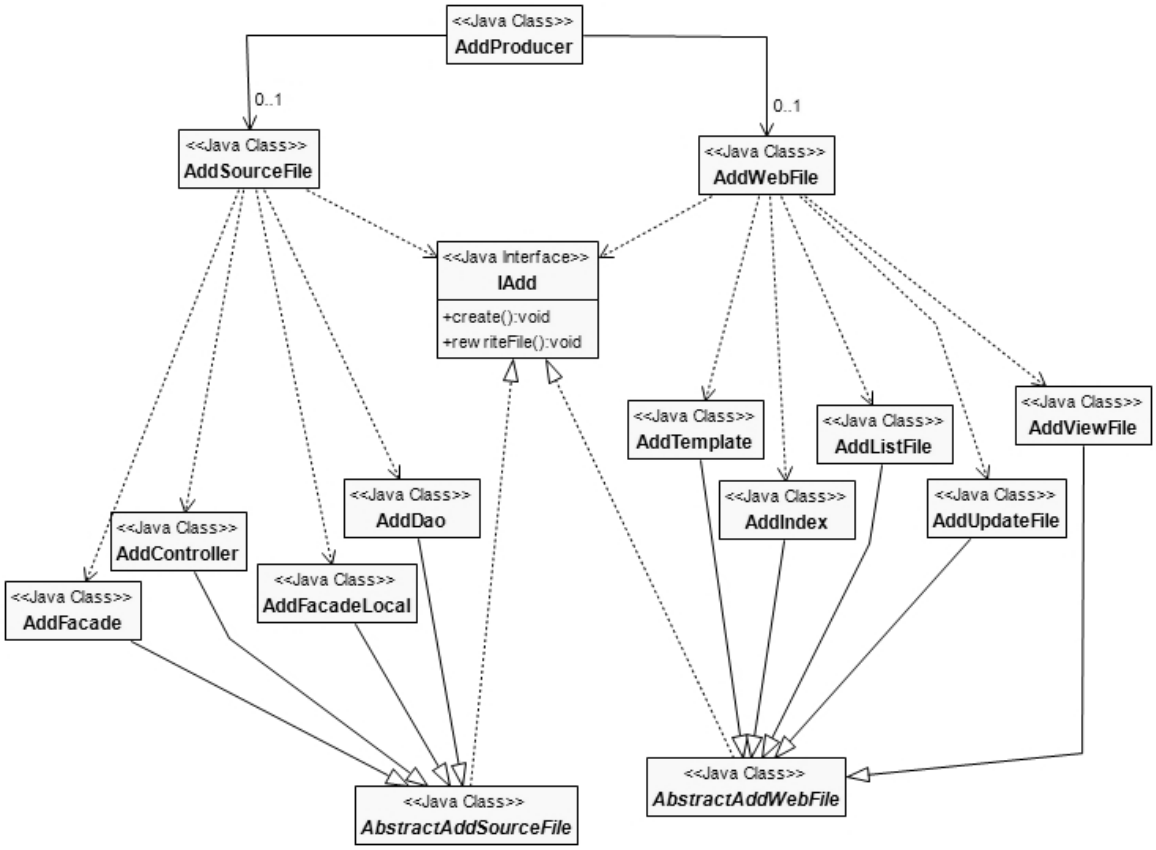
Vrstvy v samotné webové aplikaci jsou už hodně závislé na nastavení entit, jako jsou datové typy, vazby, identifikátory atd. Obě tyto vrstvy mezi sebou komunikují a předávají si informace. Aplikační vrstva se servlety a manage beanami získává a udržuje data z doménového modelu. Každý z těchto dvou typů kontroléru musí obsahovat kromě základních metod pro správu operací (vytváření instancí, mazání, atd.) pro správu složitějších typů jako jsou enumy, data, pole znaků, vazby a jiné. S každým kontrolérem entity budou spojeny všechny webové stránky této entity, jak bylo ukázáno na Obrázek č.4. Pro každou entitu se vytvoří hlavní stránka této entity se seznamem její instancí a odkazy pro další práci s instancemi. To znamená, odkaz pro vytvoření nové instance, prohlížení vybrané instance, úpravu staré instance anebo její smazání. Na každé webové stránce se bude řešit to stejné, co v kontroléru, převody datových typů na potřebný text či webovou komponentu. Aby bylo možné přistupovat k jednotlivým domovským stránkám entit,

tak je nutné vytvořit jednu hlavní kořenovou stránku s odkazy na všechny vybrané entity. Pro zjednodušení výsledného kódu a stejného vzhledu by bylo vhodné vytvořit šablonu pro jednotlivé stránky. Celá struktura webové aplikace je na Obrázek č.12, kde lze vidět závislost všech stránek jedné entity na jednom kontroléru a na šabloně.



Obrázek č.12. Struktura webových souborů a jejich závislosti na sobě

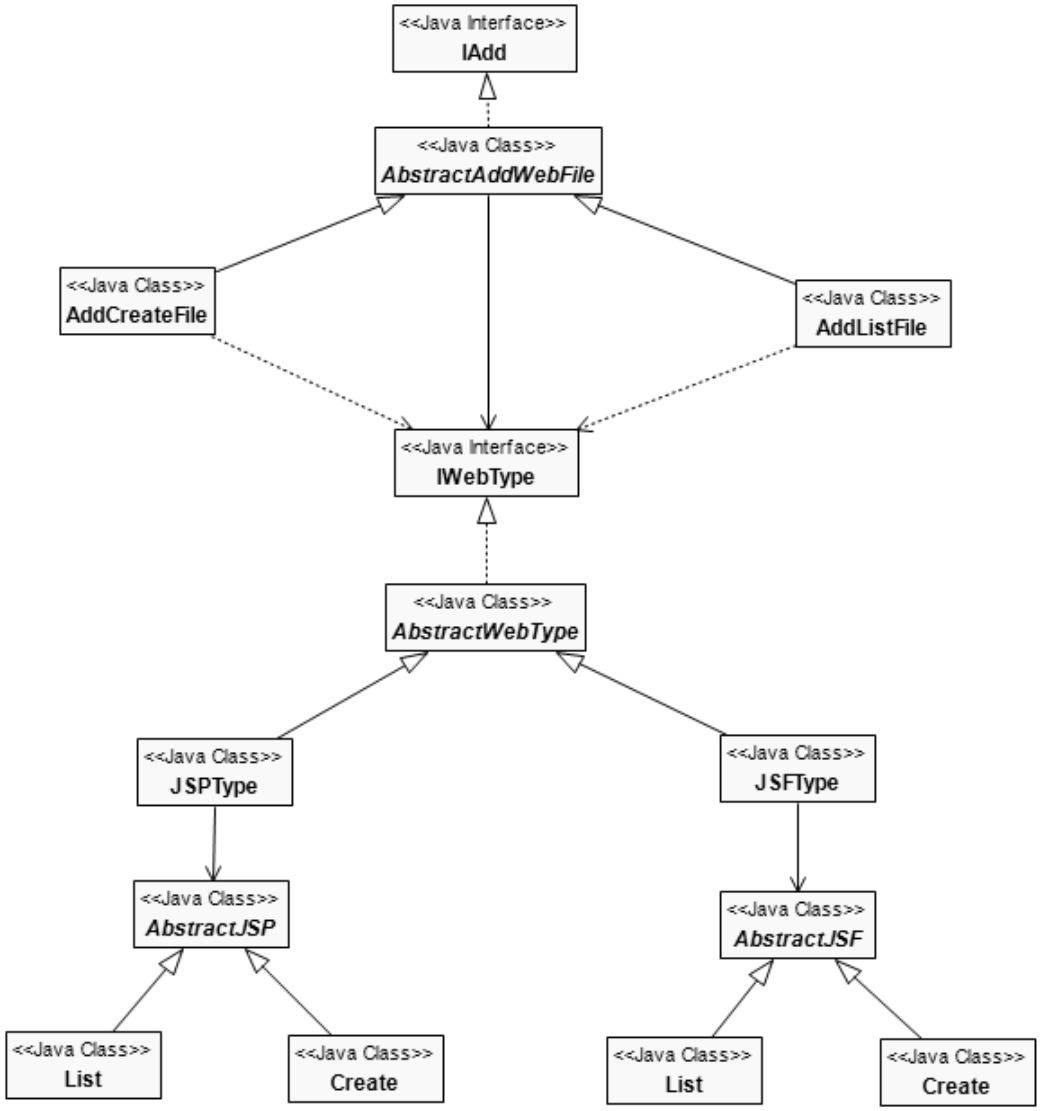
Ze souborů všech vrstev je patrné, že všechny vygenerované soubory budou Java zdrojovými soubory, až na prezentační vrstvu se svými JSP nebo JSF stránkami, které jsou založeny na XML souboru. Tím se aplikace rozpadne na dvě části, pro webové soubory a pro zdrojové soubory. Nabízí se tento problém vyřešit přes návrhový vzor „abstract factory“, kde se bude dělit kód na tyto dvě části. Všechny soubory pak budou potřebovat přijmout přibližně stejné informace jako je název, umístění, data. Proto místo rozhraní zvolím abstraktní třídy implementující toto rozhraní, které bude obsahovat metody pro všechny potomky obou abstraktních tříd, to znamená, že rozhraní bude obsahovat metody pro vytvoření a přepsání existujícího souboru a všichni potomci budou muset implementovat tyto dvě metody z rozhraní. Celý zjednodušený třídní diagram je na Obrázek č.13.



Obrázek č.13. Třídní diagram návrhu pro generování všech typů souborů

Soubory prezentační vrstvy

Vytvořený plugin bude mít možnost generovat 2 typy webových stránek, starší typ JSP a novější JSF. Oba tyto typy souborů obsahují odlišný kód, a tak je vhodné navrhnout model pro generování těchto souborů. Struktura vygenerovaných souborů by měla být stejná, takže soubory obou typů budou stejně pojmenované a musí mít stejné chování, to znamená, že se všechny soubory budou generovat stejně, budou mít pouze odlišný formát zdrojových kódů a formát uložení, což směřuje k návrhovému vzoru „factory“, aby bylo zajištěno pro oba typy stránek stejné generování a chování souborů. Aby všechny vygenerované soubory nebyly generovány jen v jediné třídě, vytvořím pro každý typ souboru jednu třídu opět se stejným rozhraním, což je opět návrhový vzor „factory“, kde každá z těchto tříd bude mít stejné metody pro výsledné vygenerování. Na Obrázek č.14 je zobrazena struktura generování vybraných dvou souborů, první pro přidávaní nových instancí, druhý pro zobrazení seznamu instancí, ostatní soubory budou strukturovány stejně. Na vrcholu je rozhraní IAdd, abstraktní třída a dva zmiňované soubory, což bylo použito v návrhu generování všech souboru na Obrázek č.13. V další fázi následuje první „factory“ rozdělující generování na 2 větve, pro JSP a JSF. Oba tyto typy se dále dělí na jednotlivé soubory přes použití „factory“, v těchto posledních souborech bude finální kód.



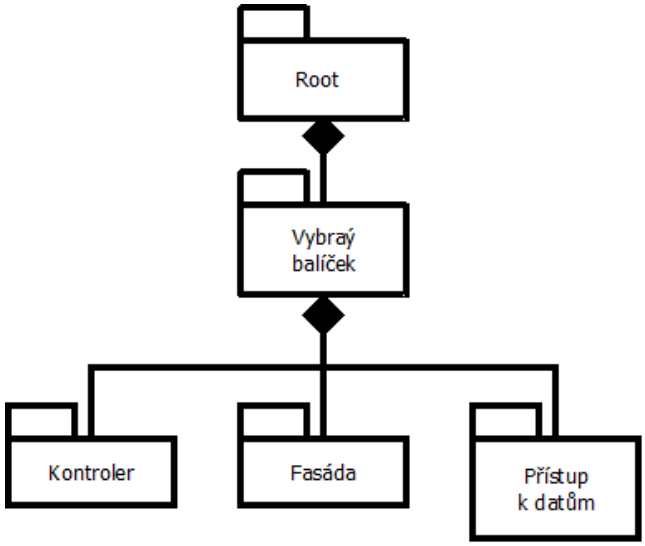
Obrázek č.14. Třídní diagram pro strukturu webových souborů

3.3.5 Rozšiřitelnost

Výsledný plugin bude nabízet na výběr ze dvou typu webových stránek, ale jak už bylo zmíněno, nemusí se zůstat jen u JSP a JSF, avšak plugin může být rozšířen o nové typy webových stránek. Z uvedené struktury na Obrázek č.14 vyplývá, že stačí přidat nový soubor implementující rozhraní IWebType pomocí něhož získá všechny potřebné metody na vygenerování všech souborů. Dále by měl být tento typ přidaný do rozhodování, který by měl být umístěný na jednom místě a přidat do wizardu, aby mohl být vybraný.

3.3.6 Uložení souborů

Jakmile jsou všechny vybrané entity zkontrolovány a po vygenerování těla daného souboru, následuje jeho uložení v projektu. Ukládaní nejde provést kdekoliv do projektu, je nutné ještě před uložením mít vytvořené složky a balíčky, které uživatel vloží ve wizardu, tyto uložiště nemusí být pouze existujícími úložišti v projektu, ale i jeho vlastní. Takže ještě před samotným ukládáním souboru je nutné vytvořit potřebné balíčky a složky. Navíc je vhodné mít vytvořené soubory nějak strukturované a ne jen uložené do jednoho balíčku. To znamená, že do balíčku přidám podbalíčky, aby odpovídaly jednotlivým vrstvám.



Obrázek č.15. Struktura balíčků v projektu

Obdobnou strukturu budou mít i adresáře pro uložení webových souborů. Jelikož ve složkách budou uloženy pouze JSF nebo JSP soubory s operacemi dané entity, tak pro každou entitu vytvořím nový adresář v adresáři, který byl vybraný uživatelem.

Po vytvoření všech balíčků a adresářů se začnou vytvářet jednotlivé soubory pro každou entitu, které budou umístěny do svého adresáře. Může se ale stát, že soubor s daným jménem bude už v adresáři existovat, například když byl plugin už v minulosti použitý se stejně nastavenými parametry. Tento problém bude nutné ošetřit, přepsat existující soubor bez povolení uživatele se nesmí stát, proto použiju, tak jak je známo v různých operačních systémech, dialogové okno s dotazem, jak se má naložit s touto kolizí stejných souborů, buď ponechat existující soubor, nebo ho přepsat novým souborem. Pokud uživatel bude znovu generovat stejné webové stránky, které obsahují třeba jen 10 entit, tak ve výsledku to znamená, že z každé entity se vygeneruje, podle výše uvedených třídních diagramů, 10 souborů, což je dohromady 100 souborů, u kterých by musel zvolit možnost přepsání nebo ponechání starých souborů. Proto by měl dialog obsahovat i možnost, aby se tato volba použila i pro další kolize, které můžou nastat. Po vygenerování

a uložení všech souborů, běh pluginu končí, a tím budou uživateli k dispozici požadované webové stránky.

3.3.7 Vazby mezi entitami a jejich zobrazení

Jak bylo výše uvedeno, existují 4 druhy vazeb, pro než je nutné rozebrat jejich zobrazení. Všechny vazby musí mít svou anotaci a u každé dvojce vazeb musí být určeno, která entita se mapuje na kterou. Ta entita, co se mapuje na druhou, nebude mít možnost vytvoření vazby, tuto možnost bude mít druhá entita ve vazbě. To nám říká, že přidávání vazeb bude moct pouze entita bez mapování.

Kromě mapování můžou entity u vazby určit, jakým způsobem se mají načítat informace k instancím druhé entity ve vazbě. Buď se tyto informace načtou ihned k instanci entity, nebo se nebudou automaticky načítat, ale jejich načtení bude nutné provést uživatel sám. Takže vazební entity u instance se budou zobrazovat pouze v případě, zdali obsahují včasné načítání informací z vazební entity (fetch=EAGER).

Poslední, co se musí u vazeb vyřešit, je jejich zobrazení a formulář pro jejich tvorbu. U vazby 1:1 se vazba vytvoří ve formuláři při tvorbě celé instance dané entity pomocí otevíracího seznamu, která nemapuje druhou entitu ve vazbě. Vazba 1:N a N:1 jsou vazby opačné. Vazba 1:N drží v databázi pouze jednu hodnotu instance druhé entity, naopak to být nemůže, z toho plyne, že vazba N:1 mapuje druhou entitu. Zobrazení bude stejné jako u vazby 1:1, neboť se bude vybírat pouze jedna instance. Zbývá vazba M:N, kde se budou určovat vazby u entity, která nebude mít mapování. Protože každá instance může držet více instancí druhé entity, tak obyčejný otevírací seznam nebude stačit. Pro tvorbu vazeb mě napadly dva různé postupy, jak na to. První, ta jednodušší je vytvoření listu s povolením výběru libovolného množství instancí. Druhou možností, asi přirozenější, je pro každou takovou vazbu mít vytvořenou novou stránku, kde bude pouze obyčejný otevírací seznam pro tvorbu nové vazby, s tím, že tohle přidávání lze provádět neomezeně.

3.3.8 Multijazyčnost webových stránek

V dnešní době je trendem mít webové stránky vytvořené ve více jazycích než jen v jednom, neboť internet je mocná zbraň, pomocí něj si může prohlížet naše webové stránky někdo na druhé stráně Země. Proto vygenerované stránky pluginem, by měly mít tuto schopnost jednoduchého nastavení multijazyčnosti.

Je důležité si stanovit způsob generování více jazyků na stránkách. Dá se říct, že existují dva různé typy, jak vytvořit multijazyčnost. Prvním je mít všechny webové soubory vytvořené tolikrát, kolik chceme mít jazyků, tento způsob je příliš těžkopádný, nepřehledný a složitý na údržbu. Druhou možností je vytvoření speciálního souboru, kde budou buď všechny slova z celé aplikace, anebo jen pro konkrétní stránku, v daném jazyce. Java podporuje tento způsob, překlady ukládá ve speciálních souborech s názvem "Soubor_jazyk_JAZYK.properties", který

vždycky obsahuje klíč slova a výstupní text v daném jazyce. Těchto souborů lze vytvořit libovolný počet, podmínkou je mít v nich stejné klíče. Tento způsob přináší kromě multijazyčnosti i výhodu jednoduššího řízení obsahu kódu.

Protože druhý způsob vytváření multijazyčnosti na stránkách je lepší, tak ho použiju. Vytvořím pouze jeden soubor, ve kterém budou všechny texty. Bude se jednat o texty, jako jsou titulky stránek, názvy tlačítek a odkazů, popisky atributů entit, apod. Formát klíčů bude text, který začne třeba názvem entity, pak bude následovat stránka, atribut a nakonec konkrétní text definující daný klíč, ukázka je níže v seznamu.

- Entita+Text
- Entita+Atribut
- Entita+Stránka+Text

3.3.9 Požadavky a omezení

Z důvodu, že vygenerované webové stránky budou Java EE aplikace, jež budou potřebovat server, tak je nutné, aby uživatel měl vytvořený soubor persistence.xml, ve kterém bude mít nastavený potřebný server s připojením do databáze, protože při generování bude nutné vložit název tohoto nastavení do kódu, bez něhož nelze komunikovat s databází přes JPA. Pokud tento soubor nebude existovat, tak se pojmenuje tohle nastavení v kódu podle názvu projektu a upozorní uživatele na tento nedostatek.

Podle definice JPA entit musí mít každá entita právě jeden identifikátor, který danou entitu odlišuje od ostatních. Základní a nejjednodušší identifikace je podle celého čísla, což bych chtěl zavést jako nezbytnou podmínku pro vygenerování stránek, pro jednolitou identifikaci všech instancí.

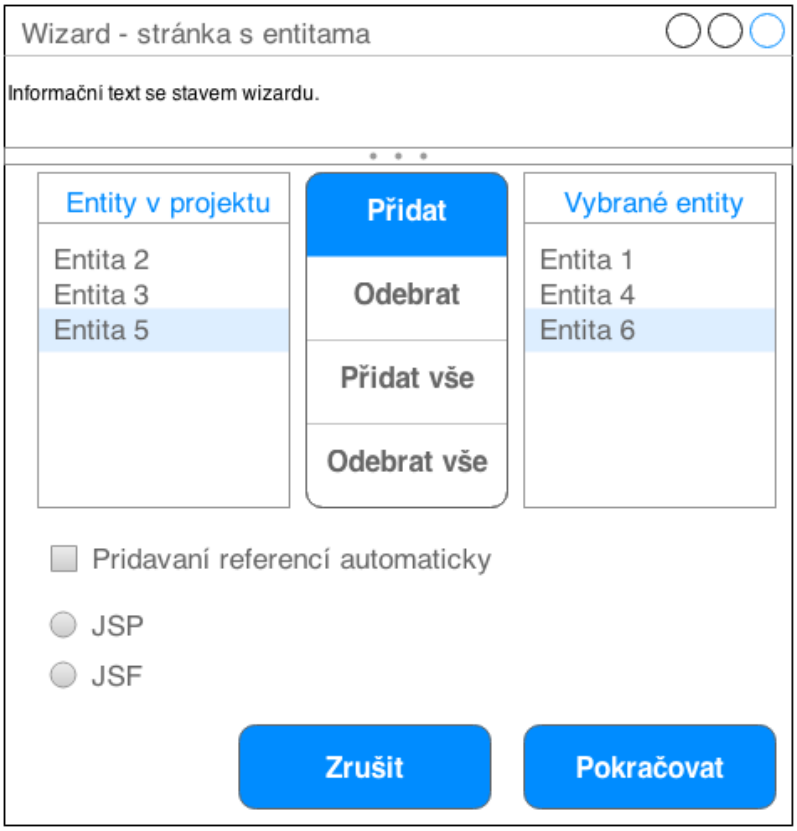
3.4 Uživatelské rozhraní wizardu

Uživatelským rozhraním wizardu bude mít na první stránce umístěny hlavní akce wizardu spojené s entitami, kde budou dva listy pro entity, první pro všechny entity v projektu a v druhém listu budou vybrané entity uživatelem. Pro přidání a odebrání entit z listu musí být vytvořeny tlačítka pro tyto akce. Pro ulehčení přidávání by mělo jít přidat i přesouvání dvojklikem. Pro entity musí wizard dále nést volbu, jestli se mají vazební entity načítat automaticky, pokud nebyly vybrány. Dále tam bude ještě výběr z typů stránek, tudíž JSP nebo JSF.

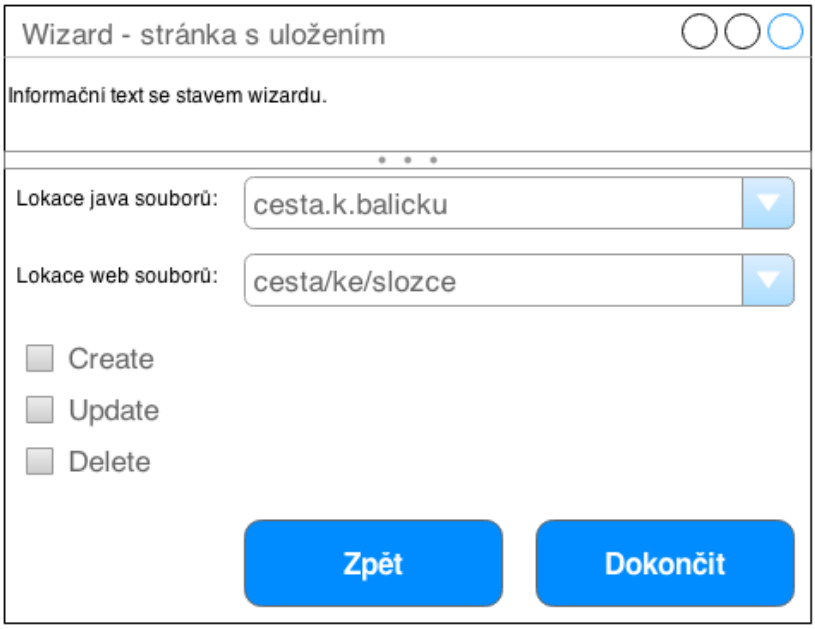
Druhá stránka wizardu by měla obsahovat otevírací seznam existujících balíčku v projektu s možností přepsání či doplnění. To stejné bude existovat i pro webové složky. Ještě chybí na výběr z možných operací, které pak půjdou na webových stránkách provádět (vytvoření,

úprava a mazání instancí), jelikož lze udělat libovolnou kombinaci, tak vhodnou komponentou jsou zaškrtávací políčka.

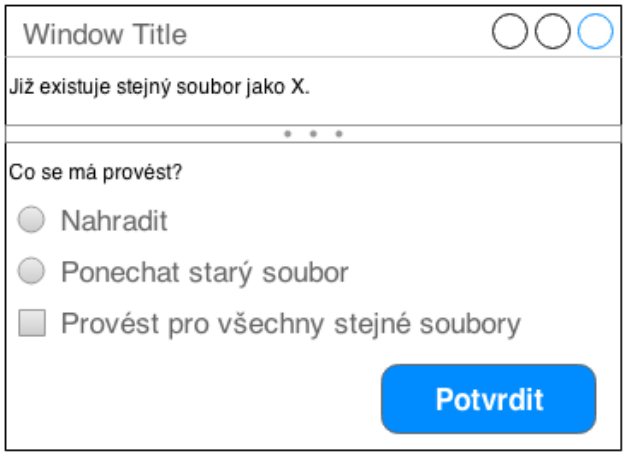
Pokud wizard objeví soubor se stejným názvem, jako má vytvořit, tak musí zobrazit dialogové okno s možnosti výběru další akce, jako je na Obrázek č.18, takže pomocí přepínače buď výběr pro přepsání souboru, anebo pro zanechání starého souboru. Okno musí obsahovat i možnost, aby vybraná akce platila i pro ostatní stejné názvy souborů.



Obrázek č.16. První stránka wizardu



Obrázek č.17. Druhá stránka wizardu



Obrázek č.18. Dialog, při existenci stejného názvu souboru

3.5 Uživatelské rozhraní webových stránek

Uživatelské rozhraní webových stránek bude nést jednoduchý vzhled, protože stránky nemají sloužit jako finální produkt, ale jen jako surový kód pro další práci s nimi. Proto i kód stránek by měl být vytvořený strašně primitivní. Kód by měl obsahovat jen prostý text s odkazy. Na hlavní stránce by měly být pouze odkazy na vybrané entity, kde u každé entity bude zobrazený její výčet instancí v jednoduché tabulce, a odkazy pro další práci, jako je jejich otevření, úprava

nebo mazání. Měl by tam být i odkaz na hlavní stránku a na vytvoření nové instance. Operace úpravy a mazání budou obsahovat jednoduchý formulář pro příjem dat všech atributů dané entity.

4 Implementace

V implementaci popíši důležité kroky při vývoji pluginu a některé důležité věci ze sekce analýzy a návrhu. Pro implementaci jsem používal vývojové prostředí Eclipse Kepler SE [22] a Eclipse Kepler [22] s vývojovým prostředím EE IDE pro vývoj webových aplikací.

4.1 Wizard

Na začátku jsem vytvořil nový projekt určený speciálně pro vývoj pluginu pro Eclipse, čímž jsem dostal xml soubor wizarda a jednu třídu, která inicializuje wizard okno, dědi z Wizard a implementuje INewWizard. Důležitým krokem je připojení pluginu k Eclipse IDE. Využiju již existujícího připojení typu command s cestou "org.eclipse.ui.newWizards", pak v XML definici pluginu existuje tag "wizard", ve kterém jsem nastavil název a přidal umístěn do složky, to způsobí, že přidá můj plugin do File->New->Other.

4.1.1 Inicializace

První co bylo nutné vyřešit, tak byla existence projektu, jež lze zjistit v metodě init, která při inicializaci přijímá 2 parametry, jeden z nich obsahuje informaci o místě, z kterého se wizard spustil. Pokud najdu zdroj, buď přes balíček, složku nebo Java třídu, tak jsem schopen z toho získat potřebný projekt, protože všechno, co patří k projektu, tak nese odkaz na projekt, což je to, co hledám.

Hned po vyhledání projektu je nutné najít nastavení projektu, ve kterém jsou definované kořenové složky pro webovou část a pro zdrojové balíčky, jež jsou nastaveny při vytváření projektu. Všechny Eclipse projekty musí obsahovat soubor XML v rootu projektu s názvem ".classpath", kde se v tagu "classpathentry" ukládá název kořenovéá složky zdrojových souborů, obvykle se používá výchozí název "src". Webový kořen je uložený v jiném souboru projektu, nachází se v souboru s názvem "org.eclipse.wst.common.component", což je taky XML soubor, ve kterém se nacházejí 2 tagy "wb-resource", kde jeden z nich je webový kořen a druhý je kořen zdrojových souborů, který jsem už vyhledal dříve.

4.1.2 Vyhledávaní entit

Další důležitou částí je vyhledávaní entit, na tohle jsem využil nástroj nazvaný JDT, který poskytuje rozhraní, které umožňuje jednoduchý přístup ke zdrojovým souborům a získávaní z něho důležité data pomocí metod, dále nám nabízí práci se soubory, jako je jejich vytváření, úprav nebo mazání. JDT poskytuje dvě API pro tuto práci, buď AST, anebo Java Model, který jsem si vybral, protože je jednodušší a smysluplnější při implementaci a rychlejší při běhu než AST. Java Model neobsahuje sice tolik informací jako AST, ale obsahuje dostatek potřebných

informací pro vytvoření mého pluginu. Navíc Java Model je definovaný přímo v jádru knihovny pro tvorbu pluginu [23].

V Java Model je každý projekt v Eclipse reprezentovaný jako Java model, který je reprezentován stromovou strukturou. Popis elementů struktury, jež jsem použil v implementaci je zobrazena v Tabulka č.1.

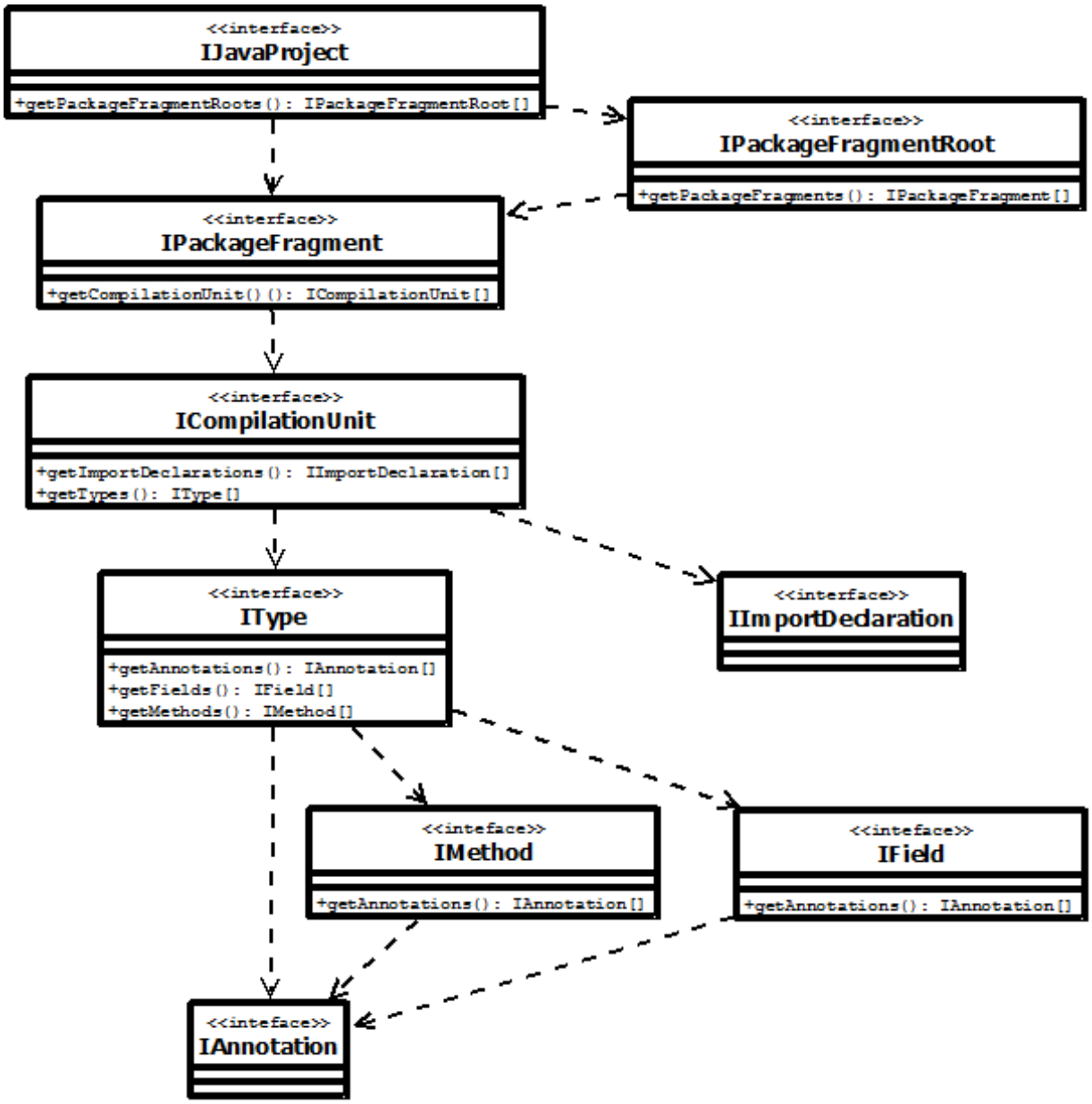
Element v projektu	**Java Model element**	**Popis**
Jva projekt	IJavaProject	Java projekt držící všechny své potomky
Root složka, balíček nebo externí knihovny	IPackageFragmentRoot	Drží zdrojové nebo binární soubory, můžou to být i složky nebo knihovny JAR nebo ZIP
Všechny balíčky	IPackageFragment	Všechny ostatní balíčky. Všechny jsou v jednom seznamu přímo pod IPackageFragmentRoot
Zdrojové soubory	ICompilationUnit	Zdrojové soubory, které jsou potomky balíčků
Importy třídy	IImportDeclaration	Všechny importy zdrojového souboru.
Typy tříd	IType	Samotná deklarace tříd, rozhraní, enumů, apod., jsou potomky zdrojových souborů
Proměnné třídy	IField	Všechny proměnné ve třídě, které jsou potomky IType
Metody třídy	IMethod	Všechny metody ve třídě, které jsou potomky IType
Anotace	IAnnotation	Anotaci může obsahovat IType, všechny proměnné a metody.

Tabulka č.1 Struktura Java Modelu

V Tabulka č.1 nejsou samozřejmé všechny možnosti, každá instance Java Modelu pak dále drží ještě plno dalších informací, pomocí niž můžeme zjistit například datový typ, typ přístupu (public, private, protected, abstract, static final), typ třídy (class, interface, abstract), předky, potomky, parametry metod, atd. Závislosti mezi jednotlivými rozhraními je ukázáno na Obrázek č.19, kde je vidět, že potomky lze získat metodou vracející pole potomků.

Pro vyhledání všech JPA entit jsem postupoval dle diagramu z Obrázek č.19, začal jsem u projektu, který jsem na začátku získal a pokračoval jsem až k třídám typu IType a u nichž jsem

prošel všechny jejich anotace. Pokud třída obsahovala anotaci @Entity, tak je JPA entitou a uložil jsem jí do pole všech entit v projektu, které se dále používá pro výběr entit uživatelem.



Obrázek č.19. Třídní diagram závislostí rozhraní Java Modelu

4.1.3 Stránky ve wizardu

Hlavní třída wizardu implementuje metodu addPages(), kde jsem přidal všechny potřebné stránky wizardu, tudíž dvě, jak bylo řečeno v sekci návrhu. Každá stránka dědí z WizardPage, který implementuje rozhraní IWizardPage pro získání potřebných metod. Komponenty na stránkách se modelují pomocí grafického frameworku SWT. Na první stránce jsem kontroloval

výběr projektu, jelikož musí být vybraný. A dále množství entit v poli entit nacházejících se v projektu. Pokud nejsou splněny obě tyto podmínky, tak se nezobrazí formulář, ale varovná hláška a zakáže se přechod na další stránku pomocí metody canFlipToNextPage(). V opačném případě se formulář zobrazí, a dále odchytávám události z formuláře, hlavně přesun entit z jednoho listu do druhého. V Metodě canFlipToNextPage hlídám počet entit vybraných uživatelem, když je počet větší jak nula, tak povolím tlačítko pro přechod na další stránku. Až uživatel klikne na toto tlačítko, tak se provede metoda getNextPage(), v které uložím všechny údaje do svého datového modelu a přejdu na další stránku.

Na druhé stránce není nic speciálního, je tam prosté nastavení úložišť souborů a výběr operací pro výsledné webové stránky. Jediné, co jsem ohlídal, je jen formát vložených úložišť, nesmí obsahovat mezery, balíčky nesmí mít lomítka, nesmí začínat tečkou apod., pokud je všechno splněno, tak protože je to poslední stránka wizardu, tak neobsahuje metody pro přechod na další stránku, ale metodu isPageComplete() pro povolení tlačítka k dokončení wizardu. Jakmile uživatel klikne na tlačítko dokončit, tak se uloží všechny údaje této druhé stránky do datového modelu a provede se metoda v hlavní třídě wizardu performFinish(), kde se spustí kontrola vybraných entit.

4.2 Kontrola

Ihned po dokončení wizarda následuje kontrola správnosti entit, a to tak, že jsem prošel postupně list entit, které uživatel vybral. Entity, jež vyhovují, jsem umístil do dalšího listu znamenající, že z těchto entit se budou generovat webové stránky. Do tohoto listu ukládám vlastní instance entit, nesoucí potřebné informace o následné tvorbě stránek, v každé instanci je pole všech zkontrolovaných atributů. U každé entity jsem řešil 3 různé případy. Nejprve jsem hledal samotnou platnost entity, tudíž vyhledal jsem její identifikátor. Začal jsem hledat v atributech entity anotaci @Id, která musela obsahovat potřebné get/set metody, pokud identifikátor obsahuje nějaký generátor, tzn. má anotaci pro generátor, nebo nemají nastavenou možnost vkládání nebo změny, tak nemusí mít set metodu. To stejné jsem provedl pro nepřímý identifikátor, kdy jsem kontroloval anotaci @Id u get metod a podle generování a nastavení jsem vyhledal set metodu.

Pak jsem kontroloval ostatní atributy entity, k nim jsou příslušné get/set metody, když ano, tak jsem zkontroloval všechny anotace dat a časů, enumů, vazeb. Pro všechny tyto typy anotací jsem vytvořil konstanty pro jejich snadnější identifikaci.

U dat jsem dále zjišťoval jejich datový typ a formát (date, time, datetime), K enumum jsem vyhledal odkaz na konkrétní enum pomocí výše zmíněného Java Modelu, existuje 8 různých možností, jak enum načítat, proto jsem musel všechny možnosti vyřešit a to přesně v daném pořadí:

- enum je deklarován s kompletní cestou na daný enum
- enum je dané v entitě

- vyhledání přes importy vedoucí přímo na samotný enum
- vyhledání přes importy balíček s hvězdičkou, kde je potomek samotný enum
- vyhledání přes importy vedoucí přímo třídu, ve které je enum
- vyhledání přes importy balíček s hvězdičkou, kde je potomek třída s enumem
- enum leží ve stejném balíku jako entita
- enum leží třídě, který je ve stejném balíku jako entita

Dále jsem řešil v plugin všechny vazby. Myšlenka je v tom, že když neexistuje druhá entita ve vazbě, tak je tento atribut s vazbou zbytečné používat, a nepoužije se pro generování webových stránek. K existenci neboli platnosti entity je nutné, aby obsahovala identifikátor. U každého atributu s vazbou jsem postupně procházel jednotlivé pole entit, začal jsem s již zkontrolovanými entitami, které říkají, že jsou platné. V případě neúspěchu jsem prošel entity, které vybral uživatel, kde jsem hledal existenci pouze jejich identifikátoru. A pokud entita stále nebyla nalezena, a byla zvolena možnost automatického referencování, tak jsem zkusil vyhledat entitu v poli všech vstupních entit, ve kterém zůstaly jen nepoužité entity. Při nalezení a zkontrolování hledané entity jsem rekurzivně prošel celou tuto entitu, abych jí mohl taky uložit a našel její případné další vazby. Problém vznikl, když vazby těchto dvou entit odkazovaly cyklicky na sebe, to jsem vyřešil, že jsem původní entitu přesunul do koncového listu, čímž jsem zamezil znovu kontrolování a nastavil jsem příznak, který určuje, že entita byla už uložená do koncového listu entit, a to aby se znovu neukládala. Pokud atribut odkazuje na sebe, to znamená, že vyhovuje a vazba je vždycky platná. Pokud jsem našel platnou vazbu, tak jsem jí uložil do pole atributů s konstantou určující danou vazbu a ještě k tomu jsem prohledal parametry anotace vazby, kde můžou být informace o typu načítání a jestli se mapuje nebo je hlavní entitou ve vazbě, obě tyto informace jsem uložil k atributu.

Poslední způsob, jak použít property, je její nepřímé volání pomocí anotace u metody, takže jsem prošel všechny metody a hledal u nich anotace. Při nazelení anotace u get metody jsem dále podle potřeby vyhledal ještě set metodu. Při vyhledání potřebných metod jsem vytvořil novou property, kde jsem nastavil, že se jedná o typ nepřímého volání, uložil jsem i odkaz na get metodu, kvůli dalším potřebným věcem, nejčastěji určení datového typu.

U všech úspěšně zkontrolovaných atributů jsem nakonec ještě prohledal všechny anotace atributu, jelikož každý atribut může mít vlastnost povinného vyplnění.

4.3 Generování zdrojových souborů

Po zkontrolování všech entit jsem získal list entit, z kterých jsou vytvořeny webové stránky. Ještě před samotným generováním souborů je nutné najít, popřípadě vytvořit potřebné balíčky.

Pro tvorbu a hledání balíčků jsem použil opět Java Model, který tohle všechno umožňuje, kde stačilo použít na PackageFragmentRoot metodu createPackageFragment s názvem nového balíčku. Všechny potřebné balíčky jsem uložil, abych do nich mohl generovat soubory. Jednotlivé soubory se ukládají do balíčku stejně jednoduše, a to pomocí metody createCompilationUnit, kde předávám název souboru a jeho zdrojový kód.

4.3.1 Soubory pro přístup k datům

Soubory pro přístup k datům se zkráceně nazývají DAO soubory a zajišťují přímou komunikaci s relační databází pro vkládání nebo zisk objektů. Jak jsem uvedl v kapitole 3.3.9, že jako identifikátory budu využívat pouze celočíselné identifikátory, a protože všechny operace v DAO využívají jako parametr nebo výstup buď objekt, anebo identifikátor, a tak jsem použil abstraktní třídu s generickým datovým typem, čímž jsem zajistil, že všechny operace, tedy metody, jsou obsaženy pouze v abstraktní třídě. Samotné třídy DAO každé entity dědí právě z této abstraktní třídy a předávají jí pouze odkaz na třídu, jak je ukázáno v Příklad č.3.

```
public class MyEntityDAO extends GenericDAO<MyEntity> {
    public MyEntityDAO() {
        super(MyEntity.class);
    }
}
```

Příklad č.3 DAO třída dědící z GenericDAO

Zajištění komunikace s databází

Komunikaci s databází zajišťuje "entity manager" pomocí anotace @PersistenceContext zajišťující objektově relační mapování. Tato anotace potřebuje název persistentní jednotky, který je umístěný v xml souboru persistence.xml v konfigurační složce "META-INF" umístěné v kořenové složce zdrojových souborů. V mém případě jsem umístil managera do abstraktní třídy DAO. Před vytvořením abstraktního souboru DAO vyhledám výše zmíněný soubor persistence.xml, kde získám, s využít DOM parser pro jednoduché vyhledání, persistentní název pomocí xPath z elementu "persistence-unit" a atributu "name". Pokud jsem nenašel tento název, tak vytvořím vlastní název z názvu projektu a na konci wizardu nechám zobrazit dialogové okno s varovnou hláškou ohledně nenalezení persistence.xml a místě, kde ve vygenerovaném kódu je tento název.

4.3.2 Fasáda

Už několikrát jsem v této práci zmiňoval, k čemu slouží fasáda, že je to rozhraní mezi webovou aplikací a doménovým modelem. Z každé třídy jsem vytvořil její vlastní fasádu. A protože se jedná o rozhraní, tak jsem vytvořil ke každé fasádě ještě jednoduché rozhraní s metodami pro práci s DAO. Do fasád implementující toho rozhraní jsem pak umístil pouhé mapování na DAO pro zaslání či zisk objektu z relační databáze.

4.3.3 Kontrolér

Kontrolér je stále zdrojovým souborem java, ale patří už do webové aplikace, tudíž uvnitř musí obsahovat rozhodnutí, který typ kontroléru má vytvořit, jestli pro JSP stránky nebo pro JSF. Oba tyto kontroléry jsou úplně odlišné, oba jsou vytvoření podle jejich obvyklé struktury, v JSP jsou to servlety s dvěma metodami pro požadavky POST a GET, kde POST přijímá data z formuláře, a tím tedy slouží pro ukládání nových instancí nebo pro jejich úpravu. GET plní ten zbytek, tzn. přechody mezi stránkami. Servlet obsahuje převody všech datových typů a v odpovědi posílá informace pro další generování stránky.

U JSF stránek se kontroléru říká ManageBean. Obsahují pro každou akci zvlášť metodu, která plní určitou funkci, třeba uložení instance, přechod mezi stránkami nebo jen na požadavek, aby poslal data webové stránce pomocí EL.

4.4 Generování webových souborů

Samotné generování webových souborů jsem vytvořil podle navržené struktury pro všechny typy webových stránek, takže pro JSF a JSP. Jelikož webové soubory už nejsou součástí Java Modelu, tak jsem byl nucen použít jiný způsob vytváření, použil jsem rozhraní z balíčku org.eclipse.core.resources, kde jsou rozhraní pro prací se zdroji (IResources), složkami (IFolder), soubory (IFile) a počátečním projektem (IProject), tyto rozhraní slouží pouze pro samotnou práci se soubory, nikoliv k vnitřnímu zisku dat jako u Java Modelu. Tento způsob zisku informací jsem v projektu použil pro všechny nezdrojové soubory, tudíž i pro získání souborů s kořeny a i pro persistence.xml.

Při tvorbě všech webových souborů jsem se snažil použít co nejméně statické textu, převážně jsem používal buď proměnné, které vycházejí z názvu entit a atributů, dále jsem často využíval statické proměnné stránek, ale taky jsem si vytvořil i vlastní statické proměnné, například pro položky, jež všechny nesou název "item", a mnoho dalších. Výstupní text na stránkách je generovaný výhradně z lokalizačního souboru, kde názvy jsou tvořeny podle výše zmíněného formátu ze sekce 3.3.8.

4.4.1 Šablony

Oba typy stránek můžou obsahovat šablonu, aby se stejný kód nemusel neustále opakovat ve všech potomcích daného typu webových stránek, a měnil se jen určitý kus stránky. Můj plugin generuje šablony pro oba typy stránek. Při vývoji nastal problém s jejich umístěním. V JSF stránkách jsem zvolil, že soubor bude uložený v uživatelem vybrané složce, ale při implementaci JSP jsem zjistil, že JSP stránky musí mít šablonu uloženou ve složce tags, která leží v nastavení webu ve složce WEB-INF z webové kořenové složky. Díky toho jsem musel v kódu kontrolovat typ stránek, kvůli umístění šablony.

4.4.2 Datové typy a jejich převody

Všechny možné datové typy na webových vycházejí z datových typů JPA entit. Podle definice JPA můžou entity obsahovat pouze typy z Tabulka č.2. V aplikaci jsem ošetřil vstup i výstup ze všech těchto datových typů. JSP stránky si dokážou při prostém zobrazení dat převést většinu typů sami, výjimkou jsou pole znaků a pole bitů, dále data a časy, avšak knihovna JSTL obsahuje konvertor pro převod data na formát podle vzoru. Pro příjem dat a jejich uložení do instance entity jsem musel všechny typy převádět z řetězce ručně na požadovaný typ, u většiny základních typů knihovny "lang" stačí provést statickou metodu "valueOf", u knihoven "math" vytvořit novou instanci s textovým parametrem. Pole znaků a bitů jsem přesouval do pole ručně znak po znaku.

S převodem datových typů v JSF stránkách je to jednodušší, všechny základní typy si aplikace dokáže tam i zpět převést sama, stačilo mi ohlídat jenom všechny pole, knihovnu "math" a datumy s časy. Pro všechny zmiňované jsem vytvořil vlastní konvertor, kde vnitřní funkce je obdobná převodu těchto typů v JSP.

Všechny datové typy jsou reprezentovány obyčejným vstupním formulářovým polem. Při špatném vložení dat jsem nekontroloval formát ani obsah textu, takže když v JSP uživatel vloží do čísla nějaký text, tak stránky vyhodí výjimku. JSF stránky si tohle řeší pro základní datové typy samy.

Datový typ	**Převod**
Text	
String	
char[]	Převod přes String.toCharArray()
Character[]	Ruční převod po jednom znaku
Čísla	
Integer, Double, Short, Long, Float, Byte, int, double, short, long, float, byte	Čísla se převádějí z textu podle TypCisla.valueOf(String)
BigInteger	Text se vkládá do konstruktoru
BigDecimal	Text se vkládá do konstruktoru
byte[]	Existuje metoda String.getBytes()
Byte[]	Ruční převod po jednom znaku
Ostatní	
Data, Calendar	Bude nutný převod vloženého data ve správném formátu na správný datový typ

Tabulka č.2 Datové typy v JPA entitách

4.4.3 Formáty dat a časů

Ani jedny z obou webových typů stránek nedokáže bez pomoci uživatele zpracovat datum a čas podle potřebného datového typu a formátu. Proto jsem musel ve vygenerovaném kódu udělat převodníky z vloženého textu na požadovaný datový typ a formát. Zvolil jsem jednotný formát, všechny formáty pro datové typy jsou v Tabulka č.3.

Datový typ	**Význam**	**Definice zápisu**	**Příklad/ Příklad se zónou**
date	datum	yyyy-MM-dd	2012-04-13 / 2012-04-13-6:00
time	čas	HH:mm:ss	21:36:55
datetime	datum a čas	yyyy-MM-dd HH:mm:ss.	2012-04-13 21:36:55 / 2012-04-13 21:36:55

Tabulka č.3 Přehled formátů pro data a časy

4.4.4 Enumy a vazby

Enumy jsou, jak bylo už zmíněno, výčtové typy, což pro znamenalo jediné, vytvořit pro ně otevírací seznam. Pro získání listu položek jsem přidal k příslušnému kontroléru dané entity metodu, která načte a převede všechny možnosti do listu. Pokud atribut může zůstat prázdný, tak na začátek listu přidám ještě prázdnou položku. V JSP posílám data pomocí parametrů. Při úpravě enumu jsem udělal počáteční výběr hodnoty podle hodnoty v instanci s pomocí podmínky z JSTL knihovny. V JSF stránkách získávám data do seznamu přes EL, při úpravě si počáteční hodnotu řeší stránky samy.

Vytváření vazby lze provést pouze u atributu bez mapování, zobrazení formuláře pro práci s vazbou se provádí stejně jako u enumů pomocí otevíracího seznamu, kde vstupní data jsou taky stejné, akorát vstupem není list, ale mapa s identifikátorem a textovým popiskem, kterou vytvářím v kontroléru. Jediná rozdílná vazba ze všech vazeb, je M:N, kdy k instanci je možné vybrat více než jednu instanci vazební entity. Proto zobrazení a způsob načítání bude odlišný. V kapitole 3.3.7 jsem řešil způsob zobrazení a práce s touto vazbou. Napadly mě dva způsoby tvorby, tak jsem vytvořil pro JSP stránky první způsob, kde je výběr vazeb v seznamu s multivýběrem. Zvolil jsem jednoduchý ukázkový kód (Příklad č.4) pro úpravu položek v tomto seznamu, kde pomocí podmínky z JSTL vyberu již vybrané instance. Jednoduše projdu všechny instance z mapy a v každé instanci vytvořím vnitřní cyklus pro uložené instance vazeb, porovnávám klíče mezi těmito dvěma instancemi, pokud jsou stejné, tak byla instance předtím vybrána, tak k volbě přidám klíčový parametr "selected=selected".

```
<select name="property" multiple="multiple">
 <option value="">---</option>
 <c:forEach var="item" items="${inputMap}">
   <option
        <c:forEach var="insideItem"
        items="${item.allItems}">
              <c:if test="${item.key==insideItem.id}">
                    selected="selected"
              </c:if>
        </c:forEach>
      value="${item.key}">
      ${item.value}
   </option>
 </c:forEach>
</select>
```

Příklad č.4 Úprava vazby M:N v seznamu s multivýběrem

U stránek JSF jsem použil druhou možnost, a to vytváření vazeb zvlášť v jiné stránce, to znamená, že jsem musel vytvořit další stránky, udělal jsem to dvou krokově, první je pouze zobrazení seznamu vazeb daného atributu, a až tam je tlačítko na stránku s formulářem na vytvoření, kde formulář obsahuje, jako v předešlých vazbách, obyčejný otevírací seznam s výběrem jedné instance.

4.5 Ostatní soubory

Na webových stránkách nejsou pouze soubory jednotlivých vrstev, jsou tam i další soubory. Jedním z nich je už zmíněný lokalizační soubor, jehož tělo postupně roste s každou entitou. U každého souboru, každé entity přidávám potřebné hodnoty do tohoto souboru.

Dalšími soubory nesouvisející s entitami jsou konvertory. Konvertory jsou java zdrojové soubory, které ale patří do modelu webové aplikace. Využívají je pouze JSF stránky. Pro vytváření jsem zvolil ten jednodušší způsob, tudíž přes Java Model. Pokud uživatel zvolí ve wizardu JSF stránky, tak všechny konvertory vytvářím automaticky bez ohledu, jestli se budou ve webových stránkách používat. Vytvářím celkem 8 konvertoru pro práci s poli, datumy a časy, a s velkými čísly.

4.6 Vygenerování pluginu

Po dokončení celé implementace je nutné plugin vygenerovat, aby mohl být nainstalován a rozšířen do dalších Eclipse IDE. Existuje několik způsobů, jak plugin vygenerovat. První z nich, je vygenerování pluginu přímo do Eclipse, ve kterém se plugin vyvíjí, to znamená, že se vygeneruje JAR soubor do složky zvané "dropins", obvykle ležící ve workspace, kde jsou

všechny pluginy wizarda. Druhá možnost je vygenerování pluginu do zvolené složky a třetí je vygenerování zip souboru. Všechny tyto tři částí se provádějí ve stejném wizardu. Tohoto wizarda spustíme pomocí File → Export → Plug-in Development → Deployable plug-ins and fragments., kde se zvolí jedna z těchto tří možností, a vybere se daný plugin. Po dokončení wizarda se vygeneruje JAR soubor do vybraného úložiště. Pro přidání vygenerovaného pluginu do Eclipse stačí vzít vygenerovaný JAR soubor a vložit ho do už zmíněné složky "dropins", pak stačí restartovat Eclipse a plugin by měl být k dispozici [24].

Tento způsob pomocí JAR souborů je pro zkušenější uživatele, existuje ještě způsob vygenerování, kde vznikne něco jakoby instalační složka, nazývá se feature, pomocí které lze provést instalace přímo v Eclipse IDE přes update managera, a to buď z lokálního disku, anebo ze serveru, třeba z webového, či souborového. Pro vygenerování feature je nutné v Eclipse vytvořit nový feature projekt přes File → New → Other... → Plug-in Development → Feature Project, kde se dále vybírají naše pluginy. Do projektu by se měl přidat soubor category.xml, z File → New → Other... → Plug-in development →Category Definition, čímž se obalí pluginy touto kategorií, v update manageru slouží jako rozpoznávací kategorie [24].

Získání samotné feature se provede přes wizarda File→ Export → Deployable features, kde jsou opět tři možnosti, do složky, do zipu nebo do Eclipse složky "dropins". V záložce "Option" se musí ještě vybrat kategorie, což je ten soubor category.xml, jehož vytváření je popsáno výše [24].

Jelikož můj výsledný plugin je určený pro uživatele, tak jsem zvolil vytvoření feature. Postupoval jsem podle uvedeného návodu. Jako výstup jsem použil uložení do složky, aby šel plugin ihned nainstalovat.

4.7 Testování

Při implementaci aplikace jsem průběžně testoval, abych kontroloval funkčnost pluginu. Testování jsem prováděl poměrně často, většinou po přidání nové vlastnosti do aplikace, abych ověřil funkčnost již otestovaných typů entit.

Největším problémem při testování, ale i při zkoušení nově naimplementovaného kódu byl čas. Přestože webové stránky v Javě potřebují Javu EE, tak na počátku vývoje jsem vyvíjel s obyčejným Eclipse SE, a to kvůli toho, že není tak náročný na počítač jako EE platforma, a kvůli spuštění pluginu v nové Eclipse aplikaci, bez nutnosti nějaké instalace, bohužel nový spuštěný Eclipse byl opět pouze SE. Snažil jsem se vyvíjet v SE co nejdéle, protože spuštění testovacího Eclipsu mi vždycky zabralo kolem 15-30 sekund. Jakmile jsem se dostal k pokročilejšímu generování funkčních webových stránek, tak jsem byl nucen přejít do platformy EE, abych testoval funkcionalitu vygenerovaného kódu. Spuštění a testování plugina jako Eclipse aplikaci mi nefungovalo jak mělo, proto jsem musel pro každou kontrolu kódu a test použít výše zmíněné nainstalování pluginu do ostrého provozu, využíval jsem přímou instalaci do složky

"dropins". Nevýhodou byl čas, jak jsem zmiňoval výše, protože každá instalace pluginu potřebuje restartování Eclipse, celková doba tohoto celého procesu byla asi 3x delší než u Eclipse SE. Samotné testování vygenerovaných webových stránek jsem prováděl na serveru JBoss AS verze 7.1.1. a databázi Apache Derby verze 10.8.2.2.

5 Závěr

Vybral jsem si tohle téma na generování webových stránek z JPA entit pro vývojové prostředí Eclipse z několika důvodů, tím prvním je, že jsem chtěl implementovat v Javě, na kterou se specializuji, dále co mě vedlo si vybrat tuto práci je určitě samotné téma, které mě docela zajímá a přišlo mi vhodné něco takového vytvořit, protože jsem v minulosti už tvořil tyto typy webových stránek a věděl jsem, jak je jejich tvorba náročná.

Před samotným započetím práce jsem tušil, jak přibližně budu postupovat, a to hlavně díky zkušenostem získaných z předešlých let. Takže jsem měl představu, jak co bude vypadat a co z čeho vznikne. Při samotné návrhu jsem hodně řešil problém se strukturou vytváření jednotlivých souborů, kterou jsem v práci rozebral v kapitole 3.3.4, chtěl jsem mít návrh co nejlépe připravený pro implementaci. Při prvotní implementaci jsem trošku bojoval s vytvořením pluginu, wizarda, a jejích umístění do Eclipse IDE. Další práce pak už probíhala podle mých představ, občas jsem narazil na menší problémy, ale ty jsem vždycky snadno vyřešil. Jediné, co mi na celé práci vadilo, bylo testování a zkoušení aktuálního kódu, a to kvůli zdlouhavému čekání. Finální verze provádí všechno, co jsem v této práci rozebral a popisoval. Na konec jsem si vyzkoušel i instalaci vlastního pluginu ze serveru.

Jelikož celá implementace byla hodně abstraktní, kdy nejde určit, co zrovna bude vstupem, tak se občas stalo, že jsem musel zpětně některé vlastnosti doimplementovat. Nejvíce, co mě mrzí, že jsem neudělal, je obecný datový typ identifikátoru, vždycky jsem identifikátory vytvářel jako cele čísla, a tak tomu bylo i v této práci, ale až ke konci práce mi došlo, že identifikátorem můžou být i texty, data, apod. Zpětně jsem to už nedodělával, jelikož identifikátory používám hojně, a to nejen v koncových stránkách a kontrolérech, ale i ve fasádě a DAO, kde jsem implementaci řešil pomocí abstraktní třídy, která počítala pouze s celým číslem. A tak to je věc, která by se měla do budoucna dodělat, aby uživatel nebyl omezený pouze na celá čísla. Věc, co souvisí s touto chybou, je práce s vazbami v JSF stránkách. Vytvořil jsem tam konvertory na převod vstupu na potřebnou instanci entity, kde vznikl problém s přenosem tohoto identifikátoru, který jsem vyřešil dost nešikovně, a to tak, že informace získávám přes metodu toString, která vrací převedený identifikátor v textové podobě, a určitě by to tak být nemělo. Na konci implementace, mě taky napadlo, že ne každému musí vyhovovat zvolený formát dat a časů, asi by bylo vhodné vytvořit nějaký seznam různých formátů, z kterých by si uživatel mohl vybrat přes otevírací seznam, ale to jsou věci, které si může programátor dodělat sám. Určitě by bylo vhodné rozšířit plugin, aby u výpisu instancí generoval i jejich stránkování, protože při větším množstvím instancí dané entity může nastat problém s pamětí počítače, nebo nepřehlednosti informací.

Zkoušel jsem hledat podobnou aplikaci na internetu a našel podobný plugin na generování JSF stránek pro vývojové prostředí NetBeans, ale kromě toho jsem nic jiného nenašel. Ani Eclipse Marketplace nic podobného nebylo, a tak doufám, že někdo můj plugin bude využívat a pomůže mu ušetřit čas při vývoji JSP a JSF stránek.

6 Literatura

1. About Eclipse (vývojové prostředí) - Portal TOL On-Line Technology. *Portal T.O.L.* [Online] Leden 2014. [Citace: 28. Březen 2014.] http://article.portal-tol.net/english-language-cs/Eclipse%20(vývojové%20prostředí).

2. *Computer Hope.* [Online] [Citace: 28. Březen 2014.] http://www.computerhope.com/jargon/p/plugin.htm.

3. **Padegaonkar, Ketan.** Contributing Actions to the Eclipse Workbench. *Eclipse.* [Online] Eclipse, 2. Leden 2007. [Citace: 29. Březen 2014.] http://www.eclipse.org/articles/Article-action-contribution/index.html.

4. Eclipse Tips - Prakash G.R.: Commands Part 1: Actions Vs Commands. *Eclipse Tips.* [Online] 2. Leden 2009. [Citace: 29. Březen 2014.] http://blog.eclipse-tips.com/2009/01/commands-part-1-actions-vs-commands.html.

5. Wizard. *TechTerms.com.* [Online] [Citace: 30. Březen 2014.] http://www.techterms.com/definition/wizard.

6. **Rouse, Margaret.** Wizard. *SearchWindowsServer.* [Online] březen 2007. [Citace: 30. březen 2014.] searchwindowsserver.techtarget.com/definition/wizard.

7. Wizard. *Eclipse documentation.* [Online] Eclipse. [Citace: 30. Březen 2014.] http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fdialogs.htm.

8. **José, Jeria, a další, a další.** JFace - Eclipsedia. *Eclipsedia.* [Online] Eclipse, 6. srpen 2010. [Citace: 31. březen 2014.] http://wiki.eclipse.org/JFace.

9. **Vogel, Lars.** SWT- Tutorial. *Vogella.* [Online] 3.1, 15. říjen 2013. [Citace: 2014. března 31.] www.vogella.com/tutorials/SWT/article.html.

10. SWT: The Standard Widget Toolkit. *Eclipse.* [Online] Eclipse, 2007. [Citace: 2014. březen 31.] www.eclipse.org/swt/.

11. **Ollé, Roman.** *Dynamické generování XML formulářů na základě XSD.* Ostrava : autor neznámý, 2012.

12. **Jindřich, Zelený a Josef, Nožička.** *COM+, COBRA, EJB.* Praha : BEN - technická literatura, 2002. str. 312. 80-7300-057-1.

13. **Písek, Slavoj.** *HTML: začínáme programovat.* Praha : Grada Publishing, a.s., 2014. str. 181. 978-80-247-5059-0.

14. **Gary, Bollinger a Bharathi, Natarajan.** *JSP - Java Server Pages.* [překl.] Soukal Vladimír a Hollay Petr. místo neznámé : Grada Publishing a.s., 2003. 80-247-0340-8.

15. JavaServlets. [Online] 1. duben 2014. [Citace: 28. duben 2014.] https://kore.fi.muni.cz/wiki/index.php/JavaServlets.

16. **Marty, Hall.** *Core Servlets and JavaServer Pages.* Upper Saddle River : Prentice Hall, 2000. str. 575. 0-13-08934904.

17. Java Server Pages. [Online] 1. duben 2014. [Citace: 28. duben 2014.] https://kore.fi.muni.cz/wiki/index.php/Java_Server_Pages.

18. **Vogel, Lars.** JSF (Java Server Faces). *Vogella.* [Online] 1.4, 26. červen 2011. [Citace: 30. duben 2014.] http://www.vogella.com/tutorials/JavaServerFaces/article.html.

19. **Peter, Purich, Raghu, Kodali a Debu, Panda.** Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide. *Oracle.* [Online] březen 2007. [Citace: 30. duben 2014.] http://docs.oracle.com/cd/B32110_01/web.1013/b28221/undejbs003.htm.

20. What is JPA Entity? *Oracle Documentation.* [Online] Oracle. [Citace: 1. dubna 2014.] http://docs.oracle.com/cd/B32110_01/web.1013/b28221/undejbs003.htm.

21. **Eric, Jendrock, a další, a další.** The Java EE 5 Tutorial. *Oracle Documentation.* [Online] Oracle, září 2010. [Citace: 1. duben 2014.] http://docs.oracle.com/javaee/5/tutorial/doc/docinfo.html.

22. Eclipse. *Eclipse.* [Online] Eclipse. http:www.eclipse.org.

23. **Vogel, Lars.** Eclipse JDT - Abstract Syntax Tree (AST) and the Java Model - Tutorial. *Vogella.* [Online] 1.6, 8. srpen 2012. [Citace: 1. květen 2014.] http://www.vogella.com/tutorials/EclipseJDT/article.html.

24. —. Extending the Eclipse IDE - Plug-in development - Tutorial. *Vogella.* [Online] 2.6, 7. únor 2014. [Citace: 2. květen 2014.] http://www.vogella.com/tutorials/EclipsePlugIn/article.html.

25. Defining a JPA Entity Class. *ObjectDB.* [Online] [Citace: 1. duben 2014.] http://www.objectdb.com/java/jpa/start/entity.

7 Přílohy

Obsah cd

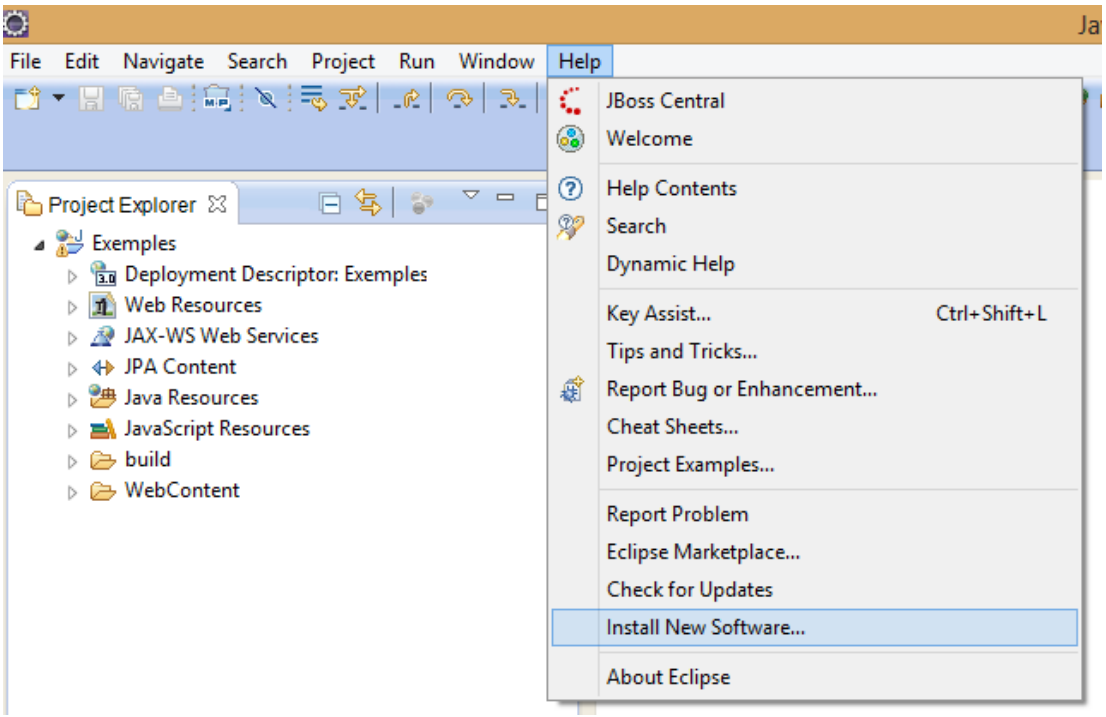
/Implementace	Zdrojové kódy aplikace
/Dokumentace	Dokumentace diplomové práce
/JavaDoc	Vygenerovaná programátorská dokumentace
/Plugin a features	Vygenerovaný plugin a feature
/Example.zip	Eclipse (64bit) s nainstalovaným pluginem a se sadou ukázkových entit

Příloha A Instalace a použití pluginu - Instalace pluginu do Eclipse a jeho použití (8 stran)

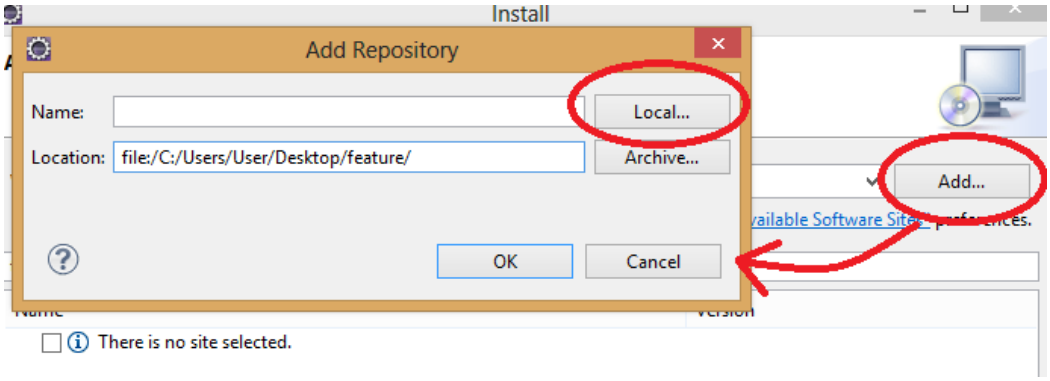
Příloha B Uživatelská příručka - Manuál k použití vygenerovaných stránek (4 strany)

Příloha A Instalace a použití pluginu

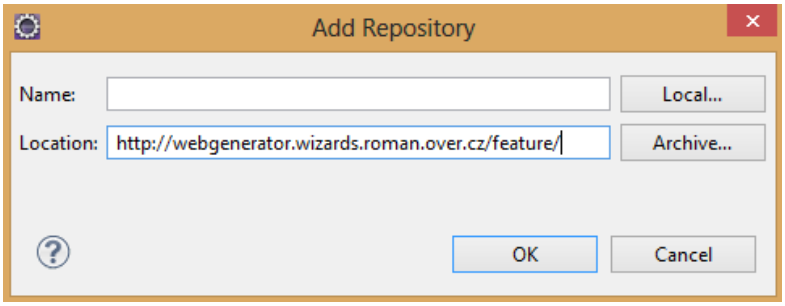
Instalace pluginu se provádí v Eclipse přes Help → Install New Software, jako je ukázáno na Obrázek č.1. Tím se nám otevře okno pro instalaci. Pomocí Add se otevře nové okno pro vložení cesty k souboru a to buď lokalně (Obrázek č.2), anebo ze serveru přes URL http://install.webgenerator.wizards.roman.over.cz/feature/ (Obrázek č.3). Tím se zobrazí náš plugin (Obrázek č.4), který vybereme a nainstalujeme, po instalaci je nutné restartovat Eclipse.



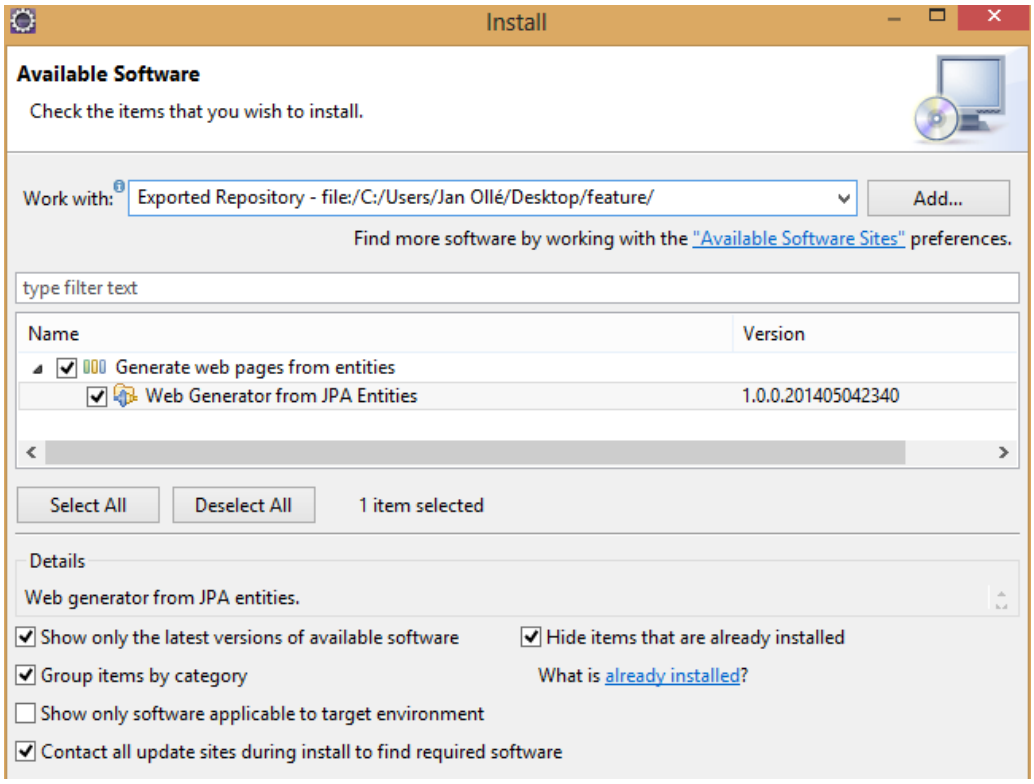
Obrázek č.1 Instalace v Eclipse



Obrázek č.2 Vložení lokace naší feature z počítače

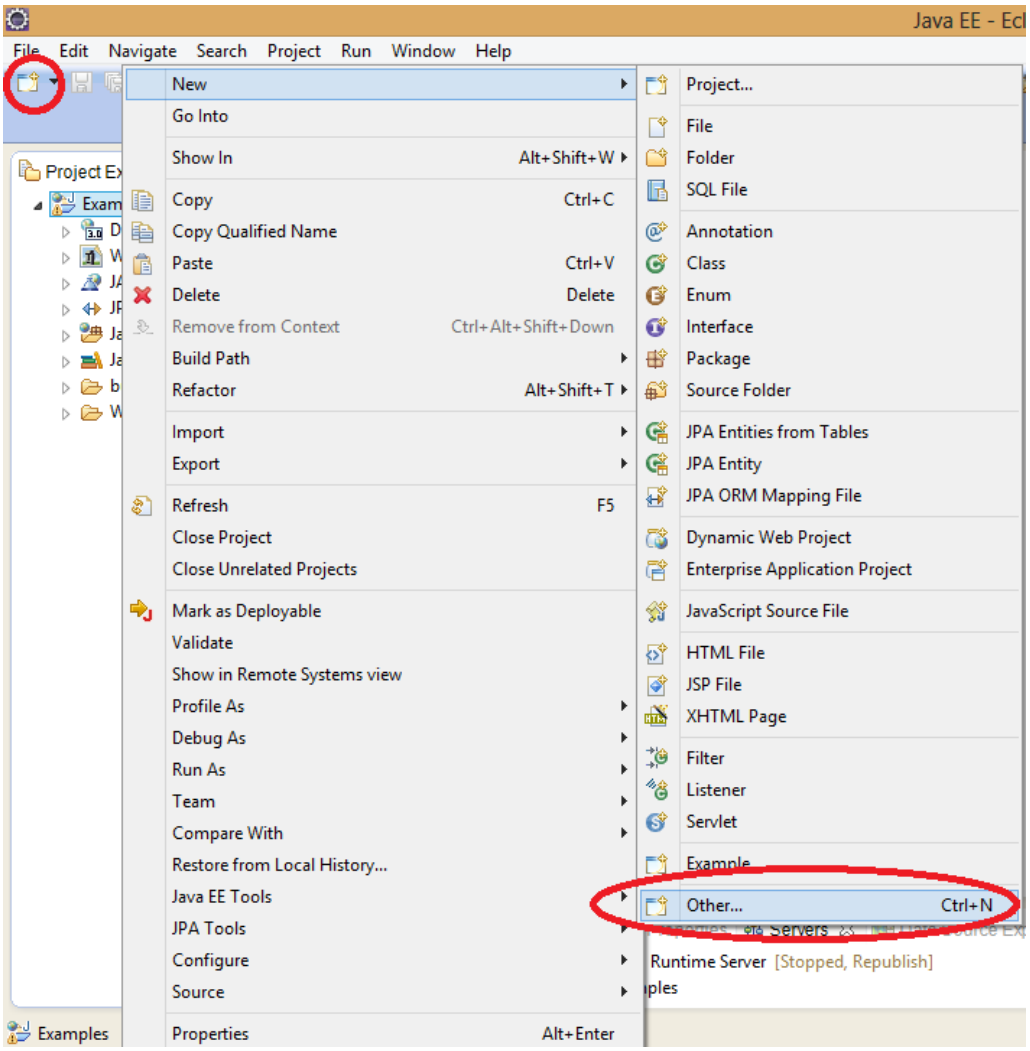


Obrázek č.3 Vložení lokace naší feature ze serveru



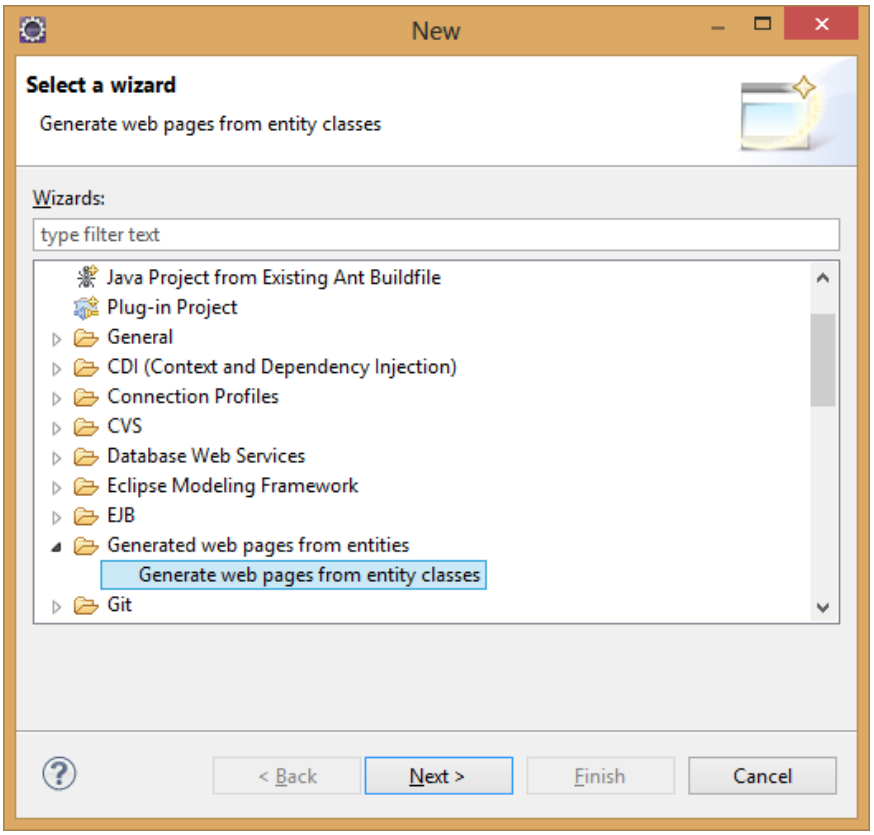
Obrázek č.4 Zobrazení pluginu a jeho výběr

Po nainstalování pluginu a restartování Eclipse, si ukážeme, jak se s pluginem pracuje. Musí se kliknout na projekt, s kterým se má pracovat, a pomocí pravého tlačítka jako je na Obrázek č.5, nebo přes tlačítko na hlavním panelu, anebo přes File → New → Other se otevře wizard.



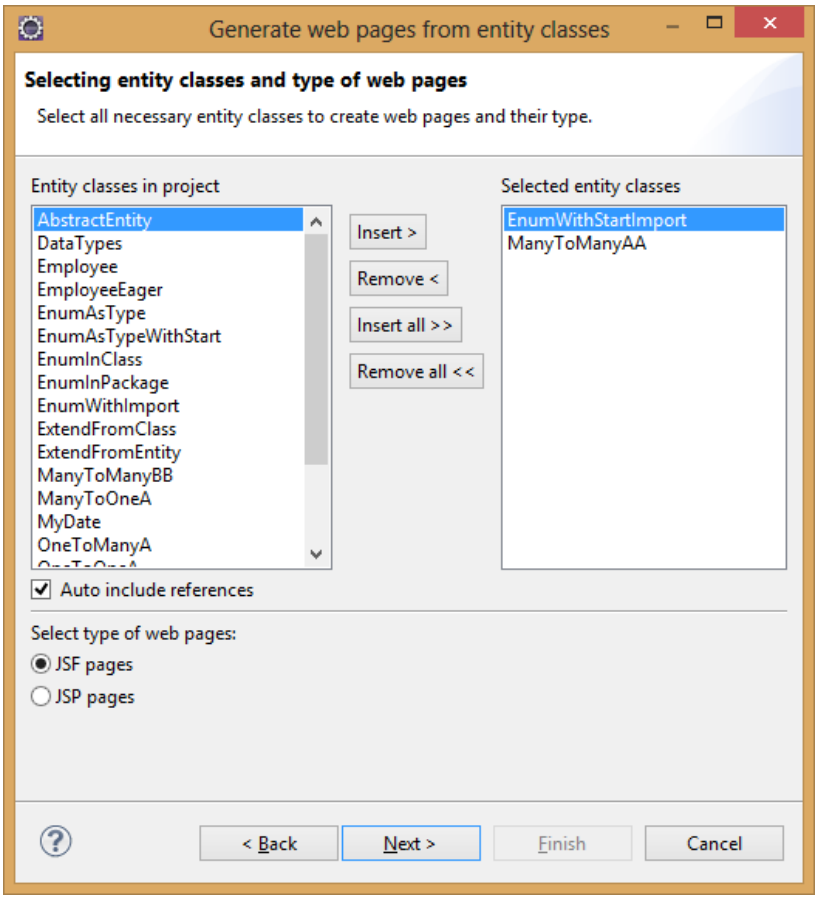
Obrázek č.5 Spuštění našeho pluginu

Tento wizard slouží pro vytváření nových věcí v Eclipse, kde je ve složce "Generate web pages from entity classes" náš plugin.



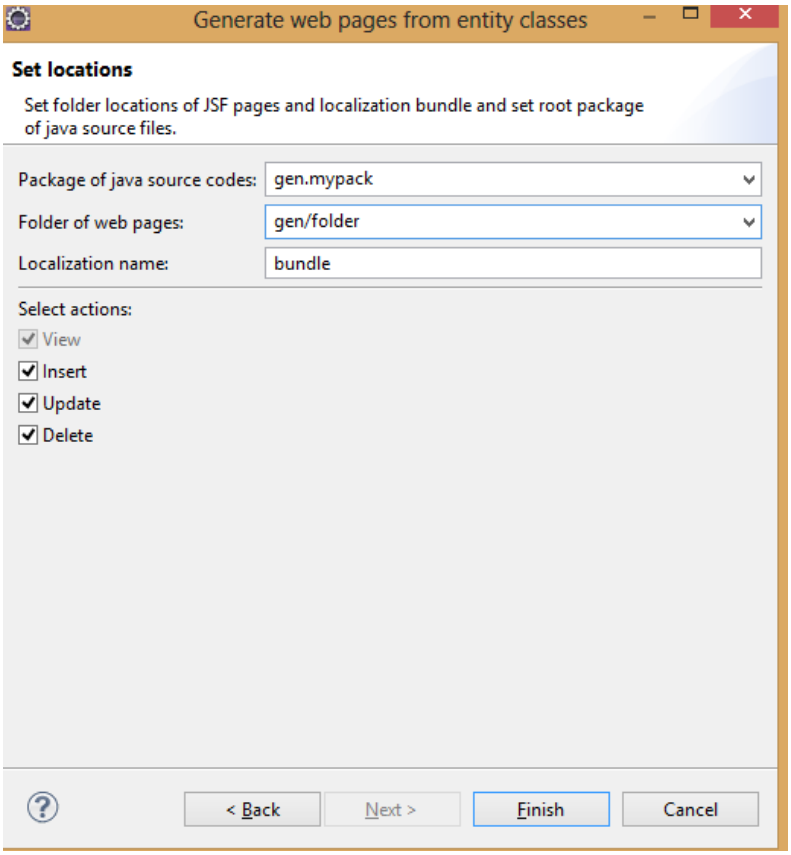
Obrázek č.6 Výběr našeho pluginu

Po výběru pluginu se spustí úvodní stránka našeho wizardu, kde je seznam všech entit ve vybraném projektu a výběr typu webových stránek. Pokud v projektu nejsou žádné entity, tak zobrazí varovný text a nepovolí jít dále k dokončení, to stejné platí, pokud nebyl vybraný žádný projekt.



Obrázek č.7 Výběr entit

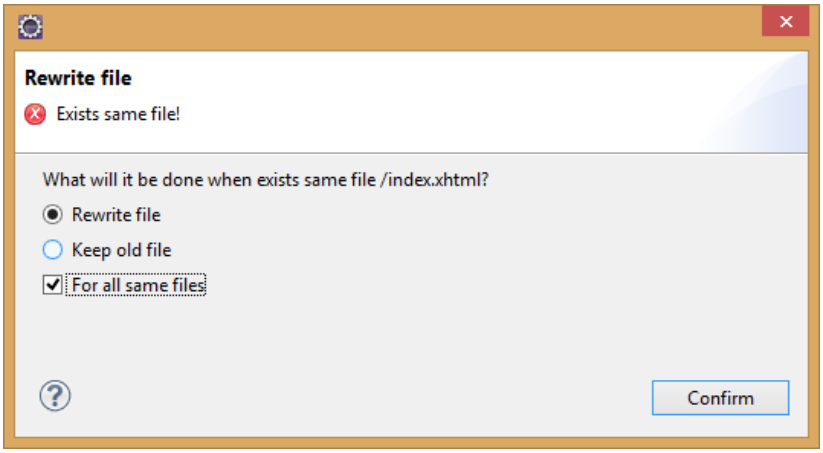
Na další stránce je práce s kořenovými cestami balíčku a webové složky, obě tyto cesty lze upravit, tzn. přidat novou cestu. Dále je zde výběr operací, které mají být na webových stránkách k dispozici. Pro ukázku vložíme balíček gen.mypack a složku gen/folder.



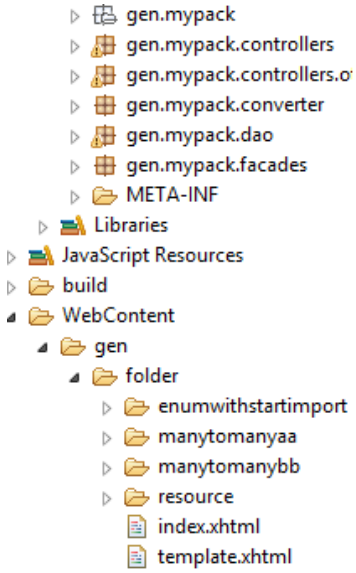
Obrázek č.8 Volba úložiště a výběr operací

Na Obrázek č.9 je dialogové okno, které se zobrazí v případě, že při generování souboru se narazí na existující soubor se stejným názvem v daném úložišti. Dialogové okno obsahuje možnosti pro ponechání starého souboru nebo tento soubor přepsat novým, a taky jestli je tato volba pro aktuální soubor nebo pro všechny další soubory se stejným názvem.

Na Obrázek č.10 je zobrazena vygenerovaná struktura všech vytvořených balíčků v balíčku "gen.mypack" a složek v složkách "gen/folder", kde jsou vytvořené další složky pro entity, které byly vybrány v prvním kroku wizardu.



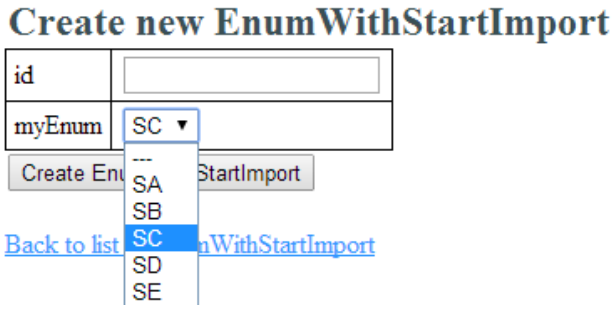
Obrázek č.9 Dialogové okno pro shodu názvu vytvářeného souboru



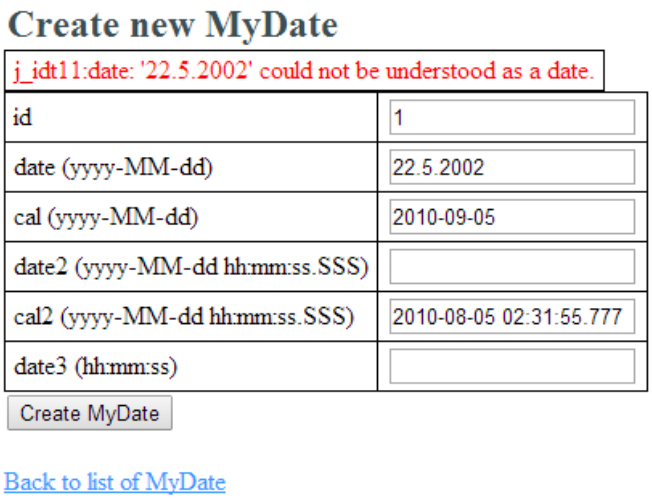
Obrázek č.10 Struktura vytvořených balíčků a složek

Příloha B Uživatelská příručka

Na kořenové stránce vygenerovaných webových stránek jsou odkazy na jednotlivé entity. Každá entita obsahuje svojí úvodní stránku, kde je výpis všech jejích instancí, jako je na Obrázek č.3. Kde každý řádek představuje jednu instanci dané entity, podle výběru uživatele nese každá instance na konci tabulky odkazy pro práci s touto instancí. Tlačítkem "create" přejdeme na stránku pro vytvoření nové instance dané entity. Na Obrázek č.1 je ukázaný jednoduchý formulář s otevíracím seznamem, pro výběr hodnoty z enumu. Na dalším Obrázek č.2 je vytváření instance, která obsahuje všechny typy dat a časů. U každého typu vstupního pole je formát jednotlivých datumů. V ukázce se jedná o stránky JSF, které si samy hlídají špatně vložené datové typy, kde v prvním datumu je vložený špatný formát, který konvertor nepřevedl, a tím stránky vyhodily chybovou hlášku.

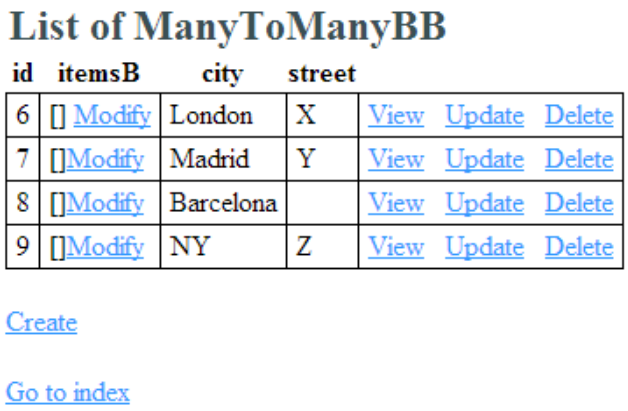


Obrázek č.1 Ukázka enumu

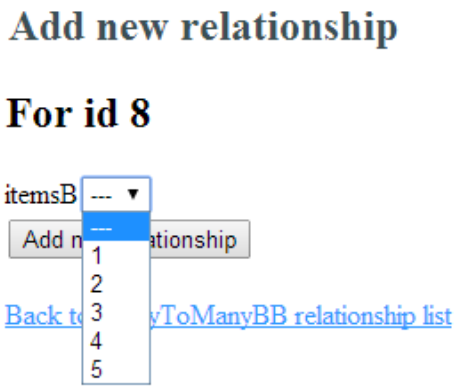


Obrázek č.2 Ukázka datumu a vyhození chyby v JSF stránkách

Obdobně jako data a časy fungují čísla, pole textů, a podobně. Dále se zaměříme na vazbu M:N, kde v JSF stránkách se vytváří zvlášť na jiné stránce než při vytváření její instance. V tabulce na Obrázek č.3jsou vytvořené 4 instancí dané entity, kde ve druhé sloupečku je vazba M:N, žádná instance zatím nemá vytvořenou žádnou vazbu. V tomto sloupečku jsou pro každou u všech instancí odkazy "modify" na úpravu a přidání nových vazeb. Tato entita může vytvářet vazby, protože nemapuje vazební entitu. Samotné vytváření je na Obrázek č.4, tvorba vazeb je stejná jako u jednoduchých vazeb nebo enumů, tedy pomocí otevíracího seznamu. Výčet instancí v této ukázce tvoří instance vazební entity, jejíž obsah je na Obrázek č.7. Počet vytvoření vazeb je v tomto případě neomezený, jelikož se jedná o vazbu M:N.



Obrázek č.3 Seznam instancí u vazby M:N s možnosti přidání nové vazby



Obrázek č.4 Přidání nové vazby M:N z instancí druhé entity z Obrázek č.7

Po návratu na seznam vazeb pomocí odkazu "Back" se nám zobrazí všechny vazby dané instance (na Obrázek č.5pro instanci s ID 8), které lze mazat. Po návratu na úvodní stránku entity lze vidět již naplněný seznam vazeb jako na Obrázek č.6, kde jsou přidané vazby i pro další instance než jen pro instanci s ID 8. Pro ID 8 lze vidět, že výčet identifikátoru vazeb se shoduje s vloženými vazbami z Obrázek č.5

List of ManyToManyBB

For id 8

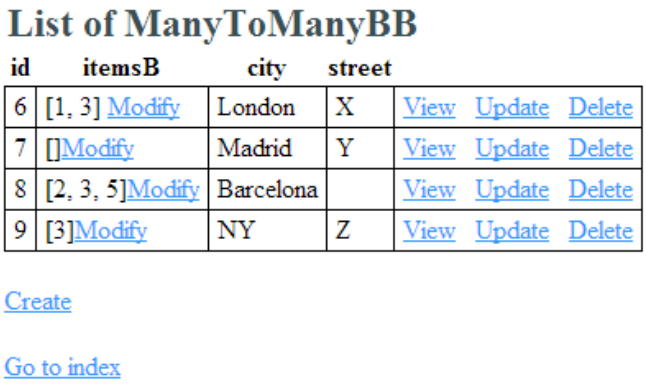
itemsB

2	Delete
3	Delete
5	Delete

Create new relationship

Back to list of ManyToManyBB

Obrázek č.5 Seznam vazeb v M:N



List of ManyToManyBB

id	itemsB	city	street			
6	[1, 3] Modify	London	X	View	Update	Delete
7	[]Modify	Madrid	Y	View	Update	Delete
8	[2, 3, 5]Modify	Barcelona		View	Update	Delete
9	[3]Modify	NY	Z	View	Update	Delete

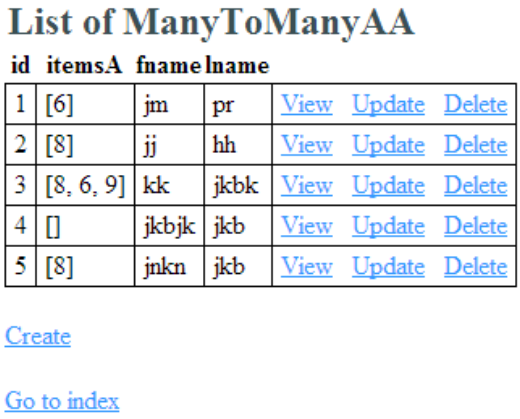
Create

Go to index

Obrázek č.6 Tabulka pro vazby M:N s vazbami z Obrázek č.5

Obrázek č.7představuje druhou vazební entitu předchozích ukázek. V druhém sloupečku se nachází identifikátory vazeb instancí vazební entity, tyto vazby odpovídají vazbám z Obrázek č.6. Jelikož tato entita mapuje předchozí, tak nemá právo vytvářet vazbu, proto sloupeček vazeb neobsahuje odkaz na vytváření a úpravu vazeb, slouží pouze na prohlížení, jedinou podmínkou je mít nastavené včasné načítání instancí.

V poslední ukázce (Obrázek č.8) je vidět stejný příklad na vazbu M:N, ale pro JSP stránky, kde se vybírají vazby ihned při vytvoření instancí pomocí multivýběru. Úplně stejně, pak probíhá úprava této instance, kde se zobrazí list s multivýběrem, ve kterém jsou vybrané hodnoty.



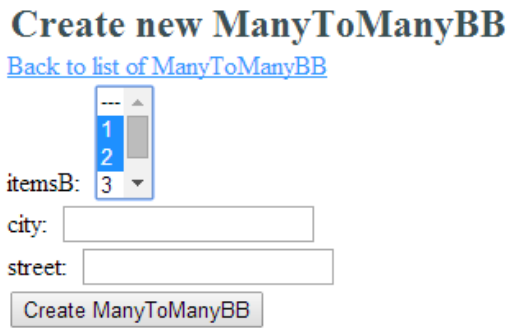
List of ManyToManyAA

id	itemsA	fname	lname			
1	[6]	jm	pr	View	Update	Delete
2	[8]	jj	hh	View	Update	Delete
3	[8, 6, 9]	kk	jkbk	View	Update	Delete
4	[]	jkbjk	jkb	View	Update	Delete
5	[8]	jnkn	jkb	View	Update	Delete

Create

Go to index

Obrázek č.7 Tabulka pro vazby M:N s vazbami z Obrázek č.5



Obrázek č.8 Vytváření instance s vazbou M:N v JSP stránkách